Oracle数据库进阶

——高可用性、性能优化和备份恢复

林树泽　李　渊　编著

15 DVD 小时

多媒体视频讲解

技术要点

- 涵盖Oracle数据库高可用性、性能优化和备份恢复的所有内容
- 超过800个典型实例，详尽的解说与过程演示
- 迈进中高级Oracle DBA一定要掌握的核心技术
- OCM考试必备

清华大学出版社

Oracle

数据库进阶

——高可用性、性能优化和备份恢复

林树泽　李渊　编著

清华大学出版社
北　京

内 容 简 介

本书内容分三大部分：高可用性、数据库优化以及数据库备份与恢复。这三部分是 Oracle DBA 必须掌握的内容，尤其是 RAC 和 Data Guard 部署在很多企业应用系统上，提供了系统的高可用性以及高可靠性，已经成为企业招聘面试的必考内容。

本书第一部分详细介绍了 RAC 和 Data Guard 的原理、架构以及安装部署技术，同时还介绍了 ASM 存储以及 Clusterware 的维护技术。第二部分详细介绍了数据库优化技术。第三部分详细介绍了 Oracle 的所有备份和恢复技术。各部分所涉及的技术都使用了大量的实例来说明。

本书将一些重要的原理、原则、个别案例制作成视频讲座，更有利于读者掌握和消化。另外，本书提供了超过 10 小时的 Oracle DBA 基础教学视频，为 Oracle 的初中级用户阅读本书提供便利。

本书面向需要进阶的初级 DBA、中级 DBA 以及准备 OCM 考试的读者。如果读者有着丰富的 DBA 经验，但对于某些原理如 Data Guard、RAC 和 ASM 等不甚了解，也可以从书中获得解答。

图书在版编目（CIP）数据

Oracle 数据库进阶：高可用性、性能优化和备份恢复/林树译，李渊编著.
–北京：清华大学出版社，2011
ISBN 978-7-302-24799-9
I. ①O…　II. ①林…　②李…　III. ①关系数据库－数据库管理系统，Oracle　IV. ①TP311.138
中国版本图书馆 CIP 数据核字（2011）第 019593 号

责任编辑：夏非彼　夏毓彦
责任校对：闫秀华
责任印制：
出版发行：清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
社 总 机：010-62770175　　**邮　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：190×260　　**印　张：**30　　**字　数：**768 千字
附光盘 1 张
版　次：　　**印　次：**
印　数：
定　价：

产品编号：

前　言

关于本书

当前市场上已经有许多 Oracle 数据库书籍，这一方面说明 Oracle 数据库产品应用的广泛性，也说明当前 Oracle DBA 职位很热门。对于喜欢钻研技术的人，Oracle 总给人带来无限的乐趣，对于为了获得更好地“钱途”的读者，Oracle 也总能如人所愿。但是，当前 Oracle 的初级 DBA 很多，而具有实战经验的 Oracle 中高级人才还是十分短缺。当前众多具有一定经验的初级 DBA，以及具有中级实战水平的 DBA，如果希望继续提高自己，往往受到理论和实践机会限制，本书出于这个考虑而编写，书中不但重视实践操作（毕竟 Oracle 对动手能力要求很高），而且给出详尽的理论解释，对于提高自己的实战能力和理论水平很有帮助。

各章内容安排

本书分三部分，这三部分都是 DBA 必须掌握的内容。第一部分是高可用性，是否掌握 RAC 和 Data Guard 已经成为一个 DBA 水平高低的指标，因为目前众多的企业用户采用这种部署来构建高可用性的 Oracle 数据库应用系统。本书的第二部分是数据库优化，数据库优化是一个复杂的系统工程，书中按照数据库组件的分类分别给出优化原则和方法，对基于 CBO 的优化给出详细的理论解释和实例分析。作为本书第三部分的备份恢复也是 DBA 必须掌握的内容，不但要掌握备份恢复工具的使用，还要重视理解备份和恢复的原理，如理解一致性备份的概念，逻辑备份和物理备份，RMAN 备份的本质等。下面是各章具体的内容安排。

第 1 章 RAC 真应用集群，本章给出一个完整的 RAC 系统的设计和实现，并对 RAC 的架构以及通信机制给出详细的理论解释。

第 2 章 ASM 自动存储管理，ASM 自动存储是 Oracle 提供的存储方案，它使用 OMF 文件格式存储数据库文件，实现文件的自动管理，是 Oracle 摆脱第三方存储的有利工具。

第 3 章 管理 Clusterware 组件，本章是对 RAC 环境维护的详细介绍，读者通过该章可以熟练掌握 VotingDisk 的维护以及 OCR 的配置和维护，详细介绍了管理 Clusterware 的指令如 srvctl、crs_stat、orcdump 等。

第 4 章 RAC 与 Data Guard，本章重点介绍了 Data Guard 的原理以及如何部署逻辑 Standby 数据库和物理 Standby 数据库，通过详细分析 Redo 的传输机制深入理解 Data Guard 的实现本质。

第 5 章 SQL 优化，本章首先介绍了 SQL 语句的执行过程，然后分析基于 CBO 的优化原理、过程以及相关的数据统计等，最后给出被动 SQL 优化和主动 SQL 优化的方法和内容。

第 6 章 Oracle 数据库实例优化，本章分析了 Oracle 数据库实例的结构，然后按照不同的结构进行优化设计，首先介绍了如何将程序及数据常驻内存，然后介绍如何优化重做日志缓冲区，共享池以及数据库高速缓存，最后介绍了如何优化 PGA。

第 7 章 I/O 及系统优化，本章从 I/O 以及操作系统两个方面介绍了优化原则，介绍了如何监控操作系统性能，以及如何理解各种操作系统指标对 Oracle 数据库系统的影响。

第 8 章 RMAN 备份与恢复数据库，RMAN 是 Oracle 推荐使用的自动化智能化的备份和恢复工具，熟练的使用和用好 RMAN 对 DBA 具有重要意义，书中详尽地介绍了 RMAN 的架构，组成以及 RMAN 备份和恢复的方法，最后给出一个实例演示 RMAN 的数据块恢复。

第 9 章 使用 EXP/IMP 工具，EXP/IMP 是 Oracle 传统的备份和恢复方法，本章介绍这种备份和恢复工具的详细用法，该工具在 Oracle10g 以及以后的版本中使用数据泵代替，但是依然支持 EXP/IMP 指令。

第 10 章 Oracle 数据泵备份与恢复，该工具相比 EXP/IMP 工具具有更好的交互性以及控制内容，本章首先引入数据泵技术的架构、优点并介绍了非常重要的目录对象概念，随后通过具体的实例演示如何使用数据泵备份和恢复数据库，如表空间、用户模式以及整个数据库。

第 11 章 用户管理的备份与恢复，本章介绍了用户管理的备份与恢复的联机与脱机方法，如何处理使用 END BACKUP 恢复表空间时的异常，然后给出具体的实例说明如何备份控制文件，整个数据库，最后通过典型的恢复示例说明如何使用用户管理的方式恢复丢失的数据文件以及恢复 NOLOGGING 的表和索引，使用重建的控制文件恢复只读表空间。

第 12 章 Oracle 闪回技术，本章是 Oracle10g 以及以后版本的数据库逻辑恢复技术，本章重点介绍了闪回删除的原理以及具体使用方法，以及闪回数据库的配置、管理和闪回数据实例，最后介绍了使用闪回数据库技术的复原点概念。

本书的特点

本书由三部分组成，分别是高可用性，数据库优化，数据库备份与恢复。本书的第一部分突出了高可用性，这部分包括 RAC 和 Data Guard，我们不但解释清楚二者的原理，更从实战示例中给出具体操作说明，这样通过原理和实践相结合，读者就可以很好的理解和掌握这部分内容。

第二部分介绍数据库优化，优化是十分复杂的系统行为，本书将优化组件进行分类分别介绍不同的优化原则以及具体的优化方法，读者再结合自己的工作经验“大胆假设，小心求证”，相信在优化之路上可以走得顺坦些。

第三部分对备份恢复工具进行了全面的介绍，并给出示例演示，这样读者在实际的工作中就可以灵活配置脚本，完成符合实际的备份方式，以及设计合理的恢复策略。对 RMAN 的介绍占用较多篇幅，希望读者重视 RMAN 的使用，并用好 RMAN。在介绍 RMAN 时本书特意说明了快速增量备份的原理以及数据块恢复的方法。对传输表空间有详尽的介绍并通过实例演示了如何使用传输表空间迁移数据，最后介绍了 Oracle10g 所特有的闪回技术，它实现了数据库的快速逻辑恢复。

本书侧重于 Oracle 数据库进阶学习，难度属于中级水平。如果读者具有扎实的 Oracle 数据库维护经验，具备 OCP 级别的理论水平，相信读者可以顺利的完成本书的学习，本书不会介绍 Oracle 的基本概念，如什么是实例、如何创建表空间等内容，所以在读本书之前请先复习 Oracle 的基础知识，或者查看本书光盘附带的超过 10 小时的 Oracle DBA 基础教学视频。

本书三个部分相对独立，各部分之间的耦合度是松散的。比如，读者需要了解书中有关高可用性的 RAC 和 Data Guard 内容，就可以可以单独学习高可用性部分；需要先掌握 ASM 存储内容，书中有一章单独介绍。每部分相对独立，读者对书中某一部分感兴趣可以单独学习。

本书读者对象

本书是一本高阶主题的 Oracle 书籍，所以适用于具有一定经验的初级 DBA，以及具有中级实战水平的 DBA。如果您正在准备 OCM 考试，相信书中的 RAC、Data Guard 以及 RMAN 部分会对您有所帮助。

本书主要由林树泽执笔。此外，参与图书编写和视频制作的还有贾东永、李华、王林、赵兵、孙明、李志国、陈晨、冯慧、徐红、杨小庆、魏刚、吴文林、周建国、张建、刘海涛、姚琳、何武、许小荣和林建新等人，在这里对他们表示感谢。

由于时间仓促，加之水平有限，书中不足之处在所难免，敬请读者批评指正。

编者

目　录

第1部分　高可用性

第 1 章　RAC 真正应用集群....................3

1.1　Oracle 为何引入 RAC....................3

1.2　RAC 概述....................4

1.3　RAC 架构详解....................6

1.4　RAC 与 Clusterware....................7

1.5　理解 RAC 的并发机制....................9

1.6　安装 RAC....................10

1.6.1　设计 RAC 应用环境....................11

1.6.2　确认安装的软件组件....................11

1.6.3　任务规划....................12

1.6.4　安装虚拟机....................13

1.6.5　在虚拟机上安装 Linux 操作系统....................17

1.6.6　配置主机....................26

1.6.7　安装 Clusterware....................39

1.6.8　安装数据库软件....................50

1.6.9　启动监听....................54

1.6.10　创建 ASM....................57

1.6.11　创建数据库....................60

1.7　本章小结....................65

第 2 章　ASM 自动存储管理....................66

2.1　Oracle 自动存储管理概述....................66

2.2　自动存储管理的优点....................67

2.3　ASM 系统架构....................68

2.4　ASM 和 CSS 集群同步服务....................69

2.5　创建 ASM 实例....................70

2.6　关闭和启动 ASM 实例....................76

2.7　理解 ASM 实例架构....................79

2.8　ASM 命令行管理工具....................80

2.9　管理 ASM 磁盘组....................83

2.9.1 使用 ASM 磁盘组管理文件的优势 83
2.9.2 创建磁盘组 84
2.9.3 向磁盘组添加磁盘 88
2.9.4 删除磁盘和磁盘组 88
2.9.5 平衡磁盘组 90
2.9.6 MOUNT 和 DISMOUNT 磁盘组 91
2.10 管理 ASM 文件 92
2.10.1 ASM 磁盘组文件名结构 92
2.10.2 ASM 磁盘组中目录管理 92
2.10.3 添加和删除别名 94
2.10.4 删除文件 95
2.10.5 使用 ASM 文件模板 95
2.11 使用 RMAN 将数据库迁移到 ASM 实例 96
2.12 管理 ASM 的数据字典视图 103
2.13 本章小结 104
第 3 章 管理 Clusterware 组件及管理指令 105
3.1 Clusterware 及其组件 105
3.2 备份和恢复 Voting Disks 106
3.3 添加和删除 Voting Disks 107
3.4 备份和恢复 OCR 109
3.5 修改 OCR 存储配置信息 112
3.6 删除 OCR 存储 114
3.7 ocrconfig 指令功能汇总 115
3.8 管理 Clusterware 指令 116
3.8.1 srvctl 指令 116
3.8.2 crs_stat 指令 121
3.8.3 onsctl 指令 126
3.8.4 crsctl 指令 130
3.8.5 ocrcheck 指令 133
3.8.6 ocrdump 指令 134
3.8.7 oifcfg 指令 135
3.8.8 olsnodes 指令 137
3.9 本章小结 139
第 4 章 RAC 与 Data Guard 140
4.1 Data Guard 是什么 140
4.2 Data Guard 配置及相关概念 140

4.3 Data Guard 服务本质......141
4.3.1 Apply 服务......142
4.3.2 Redo 应用......142
4.3.3 SQL 应用......143
4.3.4 角色转换服务......143
4.4 Data Guard 的保护模式......144
4.5 Data Guard 的优点......145
4.6 创建物理 Standby 数据库......146
4.6.1 创建物理 Standby 的前提条件......146
4.6.2 在 Primary 数据库端的操作......146
4.6.3 创建物理 Standby 数据库......151
4.7 Standby 的角色转换......155
4.7.1 物理 Standby 的 Switchover......155
4.7.2 物理 Standby 的 Failover......159
4.8 管理物理 Standby 数据库......161
4.8.1 启动 Standby 数据库......162
4.8.2 关闭 Standby 数据库......163
4.8.3 Primary 数据库结构变化的传播......163
4.8.4 自动传播数据文件和表空间的变化......163
4.8.5 手工修改数据文件和表空间的变化......165
4.8.6 重命名数据文件......167
4.8.7 添加或删除重做日志组......169
4.8.8 监控 Data Guard 数据库视图......169
4.8.9 设置 Data Guard 保护模式......172
4.9 创建逻辑 Standby 数据库......174
4.9.1 理解 SQL 应用的局限......174
4.9.2 如何唯一标识逻辑 Standby 中的表行......175
4.9.3 创建逻辑 Standby 数据库......176
4.10 逻辑 Standby 的角色转换......183
4.10.1 逻辑 Standby 的 Switchover......183
4.10.2 逻辑 Standby 的 Failover......189
4.11 管理逻辑 Standby 数据库......191
4.11.1 限制修改逻辑 Standby 数据库的对象......191
4.11.2 管理和监控逻辑 Standby 数据库视图......192
4.11.3 监控 SQL 应用过程......198
4.11.4 修改 DBA_LOGSTDBY_EVENTS 视图的相关参数......200
4.11.5 逻辑 Standby 的 DDL 操作......201

4.11.6 DBMS_LOGSTDBY.SKIP 取消同步 202
4.11.7 DBMS_LOGSTDBY.UNSKIP 恢复同步 204
4.12 深入学习 Redo 传输服务 206
4.12.1 通过 ARCn 进程来传送 Redo 207
4.12.2 LGWR 进程同步传送 Redo 210
4.12.3 LGWR 进程异步传送 Redo 210
4.13 使用 RMAN 创建 Standby 211
4.13.1 RMAN 创建 Standby 数据库的前提 212
4.13.2 RMAN 创建 Standby 数据库实例 212
4.14 RAC 环境下创建物理 Standby 217
4.15 本章小结 217

第2部分 数据库优化

第 5 章 SQL 优化 221
5.1 性能调整方法 221
5.2 SQL 查询处理过程详解 222
5.2.1 语法分析 222
5.2.2 语句优化 222
5.2.3 查询执行 222
5.3 基于成本的优化（CBO） 223
5.3.1 选择 CBO 的优化方式 223
5.3.2 优化器工作过程 224
5.4 自动统计数据 225
5.5 手工统计数据库数据 226
5.6 统计操作系统数据 230
5.7 手工统计字典数据 232
5.8 主动优化 SQL 语句 233
5.8.1 Where 谓词的注意事项 233
5.8.2 SQL 语句优化工具 234
5.8.3 使用索引 242
5.8.4 索引类型及使用时机 249
5.8.5 使用绑定变量 255
5.8.6 消除子查询优化 SQL 语句 256
5.9 被动优化 SQL 语句 258
5.9.1 使用分区表 258

5.9.2 使用表和索引压缩......259
5.9.3 保持 CBO 的稳定性......259
5.9.4 创建合适的索引......263
5.10 详解 V$SQL 视图......263
5.11 本章小结......265
第 6 章 Oracle 实例优化......266
6.1 详解 SGA 与实例优化......266
6.2 将程序常驻内存......271
6.2.1 创建软件包 DBMS_SHARED_POOL......271
6.2.2 将程序常驻内存的过程......273
6.2.3 从 DBMSPOOL 脚本理解软件包 DBMS_SHARED_POOL......275
6.3 将数据常驻内存......276
6.3.1 再论数据块缓存池......277
6.3.2 将数据常驻内存的过程......278
6.3.3 将常驻内存的程序恢复为默认缓冲池......281
6.4 优化重做日志缓冲区......282
6.4.1 深入理解重做日志缓冲区的工作机制......282
6.4.2 重做日志缓冲区相关的等待事件......284
6.4.3 设置重做日志缓冲区大小......286
6.5 优化共享池（Shared Pool）......288
6.5.1 库高速缓存......288
6.5.2 使用绑定变量......288
6.5.3 调整参数 CURSOR_SHARING 参数......291
6.5.4 设置共享池的大小......291
6.6 优化数据库高速缓存（DB Cache）......293
6.6.1 调整数据库缓冲区大小......293
6.6.2 使用缓冲池......294
6.7 优化 PGA 内存......297
6.8 本章小结......300
第 7 章 I/O 以及系统优化......301
7.1 I/O 优化......301
7.1.1 表空间 I/O 优化......301
7.1.2 数据文件 I/O 优化......303
7.1.3 表的 I/O 优化......306
7.1.4 重建索引......307
7.1.5 迁移索引到新的表空间......309

7.1.6 优化还原段 312
7.2 优化操作系统 313
7.2.1 在 WINDOWS 平台启动监控 313
7.2.2 UNIX 系统上实现性能监控 314
7.2.3 监控 CPU 的使用情况 314
7.2.4 监控设备使用情况 316
7.2.5 监控虚拟内存使用情况 317
7.3 本章小结 318

第3部分 数据库备份与恢复

第 8 章 RMAN 备份与恢复数据库 321
8.1 RMAN 概述 321
8.2 RMAN 的独特之处 321
8.3 RMAN 系统架构详解 322
8.4 快闪恢复区（flash recovery area） 323
8.4.1 修改快闪恢复区大小 323
8.4.2 解决快闪恢复区的空间不足问题 325
8.5 建立 RMAN 到数据库的连接 325
8.6 RMAN 实现脱机备份 326
8.7 RMAN 备份控制文件 328
8.8 RMAN 的相关概念与配置参数 329
8.9 RMAN 联机备份 331
8.9.1 联机备份前的准备工作 331
8.9.2 联机备份整个数据库 332
8.9.3 联机备份一个表空间 333
8.9.4 联机备份一个数据文件 334
8.10 RMAN 的增量备份 335
8.11 快速增量备份 337
8.12 创建和维护恢复目录 338
8.13 RMAN 的脚本管理 341
8.14 使用 RMAN 实现脱机备份的恢复（NOARCHIVELOG 模式） 343
8.15 使用 RMAN 实现脱机 备份的恢复（ARCHIVELOG 模式） 345
8.16 从联机热备份使用 RMAN 恢复 346
8.17 RMAN 实现数据块恢复 348
8.18 RMAN 的备份恢复验证指令 352

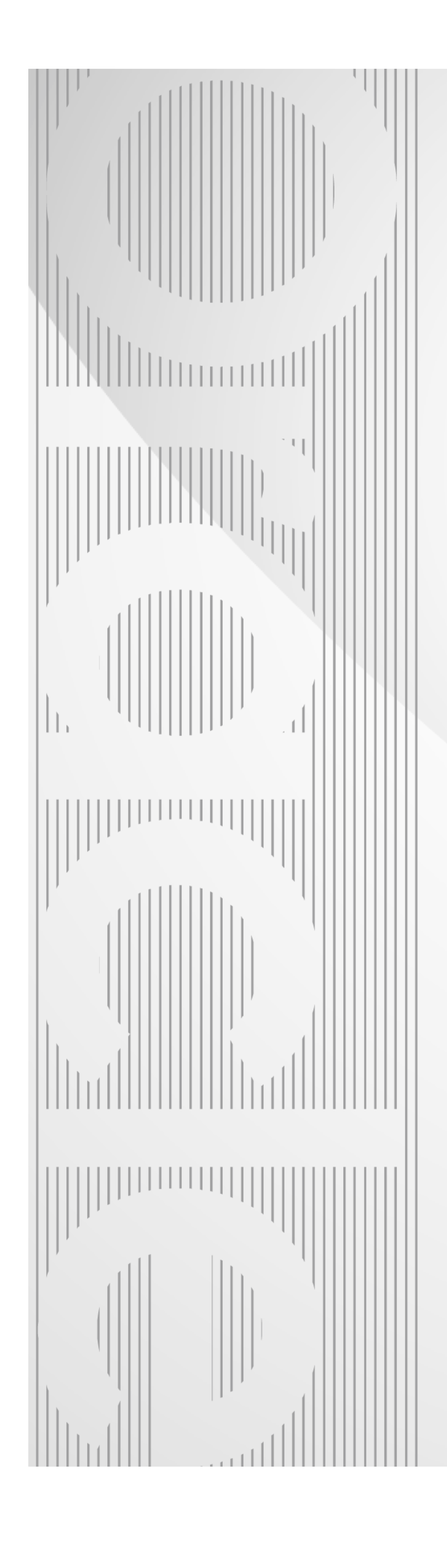
8.18.1 RMAN 的 VALIDATE BACKUPSET 指令 352
8.18.2 RMAN 的 RESTORE…VALIDATE 指令 353
8.18.3 RMAN 的 RESTORE…PREVIEW 指令 354
8.19 本章小结 354

第 9 章 EXP/IMP 及数据库备份与恢复 355
9.1 关于备份的几个概念 355
9.2 使用 EXP 指令实现逻辑备份 356
9.2.1 EXP 指令详解 356
9.2.2 不带参数的 EXP 备份 358
9.2.3 EXP 指令导出整个数据库 360
9.2.4 EXP 指令导出特定的表 362
9.2.5 EXP 指令导出指定的用户 363
9.2.6 EXP 指令导出特定的表空间 364
9.3 使用 IMP 指令实现逻辑恢复 365
9.3.1 IMP 指令详解 366
9.3.2 IMP 指令恢复整个数据库 367
9.3.3 IMP 指令恢复特定的表 368
9.3.4 IMP 指令恢复指定的用户 369
9.4 使用 EXP/IMP 实现传输表空间 371
9.4.1 理解 Big/Little Endian 371
9.4.2 传输表空间的限制 371
9.4.3 传输表空间的兼容性问题 372
9.4.4 传输表空间的自包含特性 372
9.4.5 实现传输表空间的步骤 373
9.4.6 使用 EXP/IMP 实现同平台表空间迁移 375
9.4.7 使用 EXP/IMP 实现跨平台表空间迁移（不同字节序列） 378
9.5 本章小结 382

第 10 章 Oracle 数据泵技术 383
10.1 数据泵导入导出简介 383
10.1.1 数据泵导入导出技术的结构 383
10.1.2 数据泵导入导出技术的优点 384
10.1.3 数据泵导入导出的目录对象 384
10.2 数据泵导入导出与 EXP/IMP 技术的区别 386
10.3 数据泵导出（EXPDP）数据库实例 387
10.3.1 数据泵导出的参数含义 387
10.3.2 数据泵导出数据库实例 392

10.4　数据泵导入（IMPDP）数据库实例 397
10.4.1　数据泵导入（IMPDP）概述及参数含义 397
10.4.2　数据泵导入（IMPDP）数据库实例 404
10.5　使用数据泵迁移表空间 407
10.6　本章小结 409
第 11 章　用户管理的备份与恢复 410
11.1　用户管理的脱机备份方法 410
11.2　用户管理的联机备份方法 412
11.3　备份只读表空间 416
11.4　使用...END BACKUP 指令恢复表空间备份异常 417
11.4.1　使用...END BACKUP 指令恢复表空间备份异常 417
11.4.2　使用 RECOVER DATAFILE 恢复表空间备份期间实例异常 418
11.5　备份控制文件 420
11.6　备份控制文件到 TRACE 文件 420
11.7　用户管理的全库备份 421
11.8　从用户管理的脱机（冷）备份中手工恢复 425
11.9　从联机备份中手工恢复（ARCHIVELOG 模式） 426
11.9.1　恢复数据文件 426
11.9.2　使用联机备份恢复表空间 427
11.9.3　使用脱机备份恢复整个数据库 428
11.10　用户管理的典型恢复示例 428
11.10.1　数据文件丢失（非归档模式下） 429
11.10.2　数据文件丢失（归档模式下） 430
11.10.3　使用备份的控制文件恢复新添加的数据文件 431
11.10.4　无备份条件下的数据文件重建 434
11.10.5　恢复 NOLOGGING 的表和索引 436
11.10.6　使用重建的控制文件恢复只读表空间 440
11.11　本章小结 443
第 12 章　Oracle 闪回技术 444
12.1　理解闪回级别 444
12.2　闪回删除 444
12.2.1　闪回删除原理 444
12.2.2　回收站的使用 446
12.2.3　恢复删除的表 448
12.2.4　应用 Purge 永久删除表 452
12.3　闪回数据库 454

12.3.1 闪回数据库概述 454
12.3.2 启用闪回数据库 455
12.3.3 关闭闪回数据库 458
12.3.4 闪回数据库方法 459
12.3.5 使用闪回数据库 460
12.3.6 监控闪回数据库 462
12.3.7 使用闪回数据库的限制 464
12.4 复原点技术 464
12.5 本章小结 465



·第 1 部分·

高可用性

本部分包括第 1～4 章，详细介绍了 RAC 的安装部署、RAC 架构以及 RAC 原理。对 Data Guard 不但给出详细的部署过程，还详细介绍了 Redo 传输的本质过程，这是理解 Data Guard 的核心所在；同时还介绍了 ASM 存储以及 Clusterware 的维护。

第 1 章

◂RAC真正应用集群▸

本章讲解 RAC 真正应用集群，RAC 的部署已经在中小型企业十分流行，究其原因之一是 RAC 对企业数据库提供了高可用环境，平衡数据库管理系统的访问负载；另一个原因是 RAC 的部署成本较低，部署 RAC 只需要下载免费的 Clusterware 集群件软件，即可方便地在开源系统如 Linux 上安装。本章首先从提高系统可靠性和平衡负载入手分析 RAC 的好处，接着介绍 RAC 的架构、组件及其功能和完成多节点环境下处理并发的机制，然后用较多的篇幅介绍如何安装和部署 RAC。无论是集群环境还是单实例数据库都需要"存储"，我们采用 Oracle 推荐的 ASM 机制来存储和管理集群数据库文件，所以在安装 RAC 时读者一定要十分仔细，因为一个忽视就可能造成安装错误频频，需要注意的事项我们会给出"注意"字样的提示。

1.1 Oracle为何引入RAC

一个数据库产品要想获得更大的市场份额，获得用户一致的好评最关键的因素是以较低的成本并及时地满足企业用户的需求，满足企业用户对于数据库的可靠性、稳定性和安全性的需要。其实 RAC 就是提供了一种提高数据库可靠性的解决方案。当前由于 RAC 的良好表现使得一些大中型企业倾向于使用 RAC 来提高自己数据库管理系统的可靠性和稳定性。但是由于 RAC 自身安装、配置和维护的复杂性使得 RAC 的使用环境集中在银行、电信、金融等大企业中。

随着 RAC 技术的普及，尤其是越来越多的技术人员掌握了 RAC 技术之后，更多的企业会采用 RAC 方案。

部署 RAC 的两个主要优势在于提高系统的可靠性以及平衡系统负载。基于这两点，我们作一简要分析，以给出更直观的概念。

1．RAC 如何提高系统的高可用性（HA，High Availability）

RAC 高可用性解决方案的布署如图 1-1 所示。

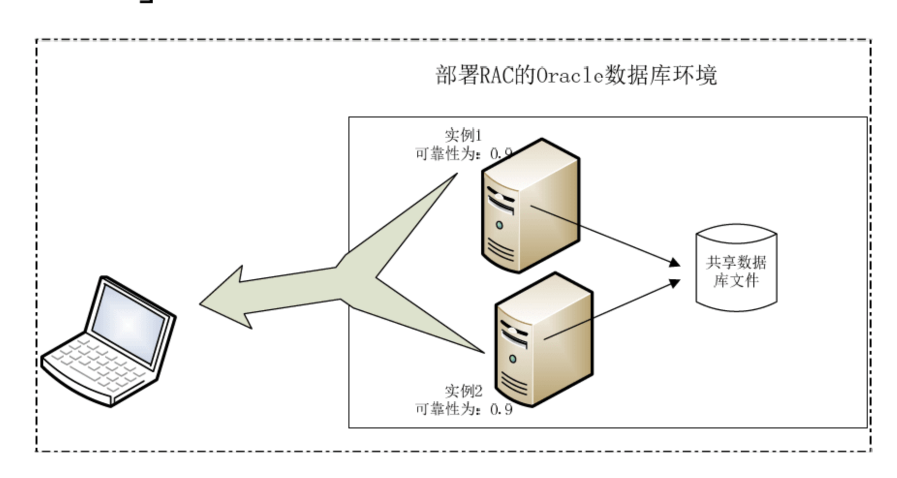


图 1-1 部署 RAC 的高可用环境

在上图中，RAC 环境提供两个实例，两个实例同时运行并向用户提供服务，这样，如果其中一个实例失败，则另一个实例可以不间断的提供用户服务。假如是单实例环境，则无法实现这样单点失效的情况。这里我们再假设每个节点的单点无故障率为 f=0.95（按照时间计算），则这样一个并联环境使得系统的可靠性大为提高，计算方式如下：

两节点系统可靠性=1-(1-0.95)×(1-0.95)=0.9975

如果是三节点的 RAC 环境，假设每个节点的单点无故障率同样为 f=0.95，我们进一步计算系统的可靠性。计算方式如下所示：

三节点系统可靠性=1-(1-0.95)×(1-0.95)×(1-0.95)=0.999875

显然这样就大大提高了系统的可靠性，部署三个节点的 RAC 环境，系统的可靠性基本上是“高枕无忧”了。这就是部署 RAC 环境可以实现系统高可用性（HA）的原因。

2．平衡系统负载（LB，Load Balance）

使用 RAC 可以提高系统的流量均衡，即在系统负载过重时，RAC 可以自动在多个节点之间平衡负载，减轻单个实例的计算压力，提高系统、数据库以及用户的交互时间。

1.2 RAC概述

RAC 到底是什么，RAC 的英文是 Real Application Cluster（真正应用集群），这个集群的最终作用就是提供系统的高可靠性（HA）以及平衡系统负载（LB），我们这里称为 RAC 集群。RAC 集群就是由多个节点组成的数据库系统，每个节点运行单个实例，这些实例之间通过特殊的机制通信以协调实例之间的并发操作以及操作同一个数据库。

我们说 RAC 集群由多个物理节点组成，是从外部物理实体角度来看的结果，那么我们使用一个 RAC 环境的数据库显然还需要理解 RAC 的逻辑组成，这样我们才可以清楚地理解 RAC 的架构以及如何安装和维护 RAC 集群。

这些逻辑结构包括：最重要的 Clusterware 存储、共享存储、网络组件以及 CRS 资源。

RAC 集群的逻辑与物理结构如图 1-2 所示。

图 1-2　RAC 集群的逻辑与物理结构

上图是一个典型的 RAC 集群环境，其中两个节点计算机安装数据库软件、Clusterware 集群软件。两台计算机分别与两个网络交换机相连，其中内网连接用于两个节点之间的协调通信，外网连接用于向用户提供数据库访问服务，两个节点共享一个数据库，二者通过部署在其上的 Cluserware 来协调对数据库的访问以及对相关资源的管理。下面我们详细介绍 RAC 集群的组成以及各个组成要素的作用。

1．集群件（Clusterware）

集群件如图 1-2 所示都安装在节点主机上，而在 RAC 集群环境下多个节点是共享存储，即操作同一个数据库的，这样就需要一个软件层来实现实例之间的协调，比如，我们知道 RAC 集群可以实现数据库系统的高可用性（HA），那么如果单个实例失败，就需系统发现这个失败而通过某种重构机制来使得系统继续对外服务，此时就需要 Clusterware 来完成这个重构过程。

Clusterware 管理相关的资源（这些资源需要预先注册），记录资源信息，监控资源状态，管理 RAC 集群中的节点状态信息。即 Clusterware 管理集群中硬件节点资源，使得这些节点对外模拟为一个虚拟的计算机，对用户屏蔽了 RAC 集群中节点的存在，节点计算机对用户透明。

2．网络要素

在 RAC 环境中涉及多个局域网络，包括内网、外网以及存储网络。其中，内网用于内部连接，这个连接实现不同节点上实例之间的协调通信，如用户 CacheFusion 传递数据块等操作。外网用户对外服务，如用户查询数据请求等操作。存储网络通过高速交换设备连接到共享存储，实现 RAC 集群中数据库的共享。

3．共享存储

我们已经知道 RAC 集群是具有多个节点的物理环境，每个节点运行单个实例来访问数据库，响应用户的请求，为了防止多个实例数据操作的不一致性，所以要求多个节点共享存储。

这样在 RAC 集群中需要将数据文件、控制文件以及日志文件存储在共享存储介质上，保证整

个 RAC 环境下只有一个操作数据库。

4．CRS 资源服务

Clusterware 用于管理 RAC 集群中多个节点的各种资源，如与数据库相关的监听器状态、实例状态，与节点相关的 VIP、GSD 以及 ONS（这三个资源应用在安装完 Clusterware 后使用 VIPCA 来启动），这些都是 CRS 管理的资源。这些资源信息会存储在 OCR 磁盘上，整个 RAC 环境只有一份资源信息，这样通过 Clusterware 中的相关进程监控 OCR 磁盘中资源信息，从而实现监控这些资源的状态的目的。

1.3 RAC架构详解

如果读者已经具备一定的单实例数据库使用经验，应该对单实例数据库的架构比较了解，而 RAC 的架构是多实例的数据库应用环境，由于需要处理并发操作，协调多个实例之间的通信，显然在架构上有区别于单实例的地方。这些不同主要体现在 RAC 架构中多了一个 GRD 内存区以及附属的多个后台进程和部分数据库文件。

从宏观上 RAC 的架构由 SGA、后台进程以及相关数据库文件组成，图 1-3 是两个实例组成的一个 RAC 架构。

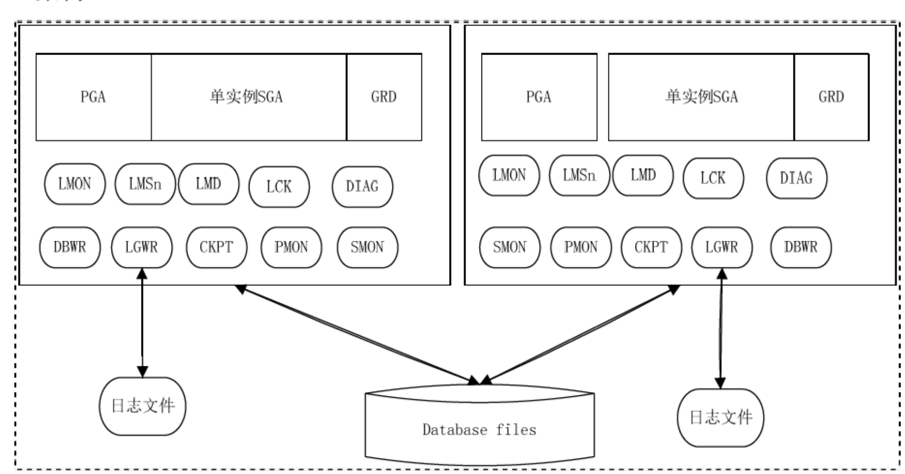


图 1-3　双实例 RAC 架构组成图

如果读者已经熟悉了单实例的 Oracle 数据库架构，想必对上图也不会陌生。整体上 RAC 架构包括 SGA 区以及各种后台进程，整个 RAC 环境中只有一个数据库，这个是多个节点共享的，读者可以想象，RAC 环境中的多个节点同时为用户提供服务，就涉及到数据的并发问题，如何控制并发行为而不导致用户操作的混乱是 RAC 需要解决的，所以 RAC 环境中需要不同于单实例结构的后台进程，完成诸如协调多实例之间数据访问操作、实例间数据传输之类的操作，所以就需要相应的后台进程来支持这种行为。下面我们依次介绍 RAC 各组件及其作用(不包含单实例中的组件)。

1．GRD 的作用

GRD（Global Resource Directory）是“全局资源目录”的意思，在 RAC 集群环境下每个节点的实例中都有一个 GRD 内存区，该区域用来存储同一个数据库在不同节点上的分布，即多个实例在并发操作一个数据块时，将该数据块存储在各自实例的 GRD 内存区中。

RAC 通过某种机制可以监控每个实例上相同数据块的当前状态，以协调多个节点对同一个数据块的并发操作。

2．LMON 进程

在 RAC 集群环境中各个节点的实例之间会定期通信，目的是检查各自当前状态，如对方实例是否异常等信息，这种在节点之间协调通信以完成健康检查的任务就是由 LMON 进程负责。

3．LMD 进程

我们已经知道 CacheFusion 的 GES（Global Enqueue Service）服务，该服务的作用是在节点之间协调对同一数据块的访问次序，进程 LMD 就是提供 GES 服务。

4．LCK 进程

顾名思义 LCK 进程就是锁进程，它管理在集群中对同一数据块访问的锁管理。

5．LMSn 进程

我们已经知道 CacheFusion 的 GCS 服务，该服务的作用是在节点之间拷贝数据块，而 LMSn 进程就是提供 GCS 服务。

6．DIAG 进程

这是一个“日志”进程，将记录集群的健康状态，并记录实例错误时的诊断信息。

7．参数文件和日志文件的存储。

由于在集群环境下，参数文件和日志文件必须在所有节点的实例之间共享，要求他们放在共享存储设备上，从而使得多个节点都可以访问到，以防止单个节点修改而其他节点“一无所知”的情况。

1.4　RAC与Clusterware

本节我们讨论 RAC 和 Clusterware 的关系，以及 Clusterware 的组成，这样读者就可以更清楚地理解 Clusterware 在 RAC 集群环境中的作用，通过学习 Clusterware 的组成就可以更清楚地明白 Clusterware 是如何完成自身功能的。

1．理解 RAC 和 Clusterware 的区别

RAC 的全名为 Real Application Clusters，翻译成中文是“真正应用集群”的意思。而 Clusterware 翻译成中文是“集群件”的意思。初学的读者往往会弄不清楚这两个名词的区别，这里我们就给出一个详细的介绍。

首先 RAC 是一个集群环境，它使得数据库服务器之间通过称为集群件的软件一起协调工作，

该集群环境提供了“实例”对“数据库”之间的多对一的关系，即多个数据库实例对应一个物理数据库。多个实例之间的工作通过 Clusterware 来协调，共同操作一个物理数据库。这个多实例的环境与单实例环境相比，提高了系统的吞吐量和可扩展性，提供了系统的可靠性和负载均衡。

其次 Clusterware 是一种集群件产品，它负责管理 RAC 的节点硬件资源，管理各种应用资源，并为集群数据库提供基础服务，二者的关系如图 1-4 所示。

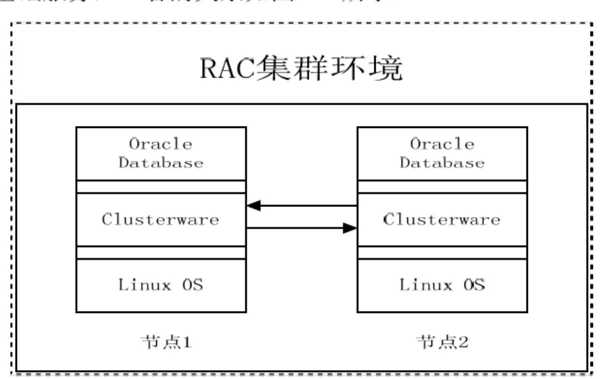


图 1-4　RAC 集群与 Clusterware 的关系

2．Clusterware 集群件的组成

Oracle Clusterware 集群件由 OCR、Voting Disk、后台进程以及网络组件组成。下面我们分别介绍这些 Clusterware 组件。

（1）OCR

OCR 是一个磁盘文件，该文件存储在裸设备上，它的作用是存储 RAC 集群节点的配置信息，因为在整个环境中只有一个存储配置的磁盘文件，所以实现了对集群配置的同步修改。

（2）Voting Disk

该文件用于存储节点状态。一旦某个节点失效，通过“投票”算法将失效的节点踢出集群。因为该文件是共享存储到裸设备，每个节点都可以访问到，通过存储的节点状态，如投票结果就可以踢出失效的节点，重构一个健康的集群。

（3）OCCSD 后台进程

Clusterware 集群件提供 CSS（Cluster Synchronization Services）集群同步服务。该服务通过某种机制来判断集群中的节点是否“活着”，监控节点健康状态。而 OCSSD 进程就是完成 CSS 服务的。

（4）CRSD 后台进程

在 RAC 集群环境中，Clusterware 集群件要管理和监控注册到 OCR 中的各种资源，如 GSD、VIP、ONS 以及 Listener 等应用资源，而 CRSD 进程就是负责监控这些资源，并在这些资源故障时提供系统的高可用性。

在 RAC 环境下，这些资源会记录在 OCR 中，而 CRSD 进程就读取 OCR 中存储的资源状态信息来管理资源，如监控资源的运行状态，何时以及如何重启或关闭这些注册的资源。

（5）其他 Clusterware 进程

其他 Clusterware 进程包括 EVMD 和 RACGIMON 进程。EVMD 进程负责分发 CRS 产生的事件（event），RACGIMON 进程负责检查数据库的状态。

（6）网络设置

RAC 集群环境中的每个节点至少需要两块网卡，一块称为 Public 网卡，配置的 IP 称为 Public IP，一块称为 Private 网卡，配置的 IP 称为 Private IP。前者用于连接外网，这样就可以通过外网向用户提供服务；后者用于内部连接，这样实现节点实例之间内部通信，如实现心跳（Heartbeat）以及内存融合（CacheFusion）等通信。

在安装 Cluserware 的过程中，还需要配置一个 Virtual IP，该地址会绑定到 Public 网卡上，以实现 RAC 集群的 Failover。

1.5　理解RAC的并发机制

并发是 RAC 环境下十分频繁的一类操作。既然有多个实例同时提供服务，并且是共享一个数据库，这样处理并发操作就体现了 RAC 自身的特色。本节我们讨论 Oracle 如何在 RAC 集群环境中实现数据的并发操作。

我们知道在任何一个软件系统中，无论实现什么功能最终都是通过某种软件组件来实现，在软件运行时就是以“进程”的方式实现。在 RAC 集群环境中，Oracle 设计了 Cache Fusion（内存融合）把多个节点中实例的 SGA 虚拟成一个大的 SGA，从而使得多个节点 SGA 对用户透明，而 Cache Fusion（内存融合）的两个主要进程就是 GCS 和 GES，其中 GCS 实现实例之间数据块的拷贝传递，GES 负责管理锁（如实例的进程只有获得某种数据块的锁后才可以操作数据，获得锁就可以使用数据块）。

下面我们解释 GCS 实现实例之间数据块的拷贝传递。当某个数据块已经保存在某个节点的数据库高速缓存中时，如果另一个实例需要读取这个数据块，此时该实例不会再请求从磁盘中读取该数据块，而是通过 Cache Fusion（内存融合技术）内部的私有连接（Private IP）传递该数据块给需要它的实例，这样对用户而言 Cache Fusion 就把多个实例的数据库缓冲区虚拟成一个数据库缓冲区。实现了 SGA 对用户的透明。

图 1-5 所示就是节点的 SGA 之间通过 Cache Fusion 的 GCS 进程拷贝数据块的过程，而对用户而言就屏蔽了多个实例的存在。

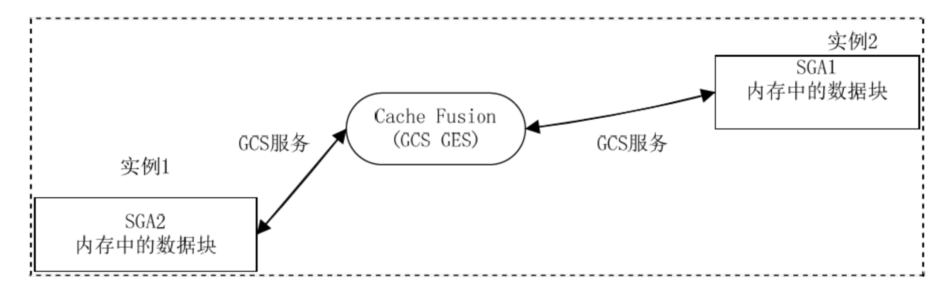


图 1-5　Cache Fusion 工作机制

为了更好地理解 RAC 如何实现对资源的并发控制，我们首先需要理解两类资源和两类锁的概念。

在单实例的数据库环境中，锁分为 local lock 和 latch，在单机上实现对数据块的并发访问；而在 RAC 集群环境下，一个数据块可能分布在不同的节点上，并且数据块处于不同的状态，那么如何实现节点之间的协调就需要其他类型的锁，即 PCM_Lock 和 Non_PCM Lock，这两种锁对应管理两种资源，即 Cache Fusion 资源和 Non_CacheFusion 资源。要通过锁管理这两种资源，在 RAC 环境下还需要一个称为锁管理器 DML 的组件，DML 类似于一个仲裁机构，决定是否允许节点上用户对某个数据块的访问请求。

首先看看什么是 Cache Fusion 资源，其实 Cache Fusion 资源就是指数据块资源，如索引块、普通表数据存储的数据块、还原段存储的数据块等，而其他非数据块资源就是 Non_CacheFusion 资源，如数据库缓存、数据文件、参数文件等。

PCM_Lock 用于协调节点之间对数据块的并发控制和访问，大致的流程如图 1-6 所示。

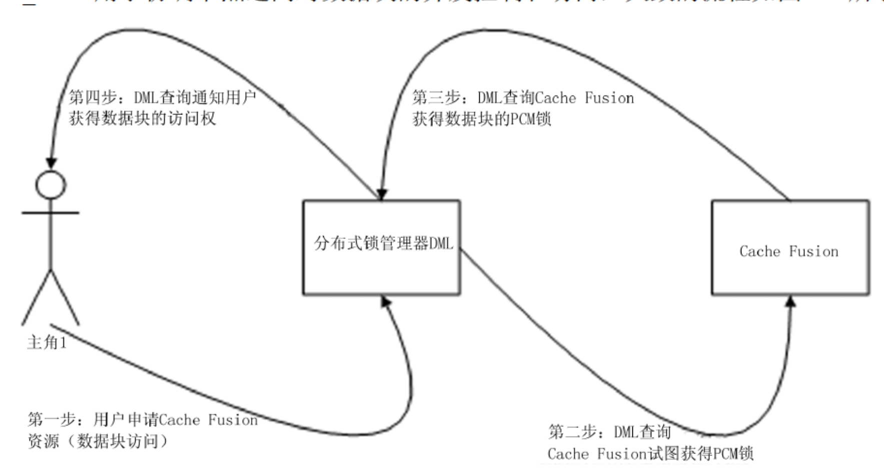


图 1-6　RAC 集群实现并发机制过程

由于在 RAC 集群环境下一个数据块有多个版本，那么记录这些数据块的版本信息、当前状态以及节点分布就是非常重要的一个行为，而 Cache Fusion 就实现了上述行为，使得 DML 可以通过查看数据块的节点分布以及当前状态等信息，来判断是否为用户 PCM_Lock 而使得用户获得对该数据块的使用权。

1.6　安装RAC

本节是本章中内容最多的一节，没有太多的理论分析，都是“实打实”的具体操作，也就是工程师在部署软件初期的“体力劳动”，但是这个劳动是值得的。如果这个过程部署顺利且系统稳定，那么对于将来的维护是十分有利的。何况 RAC 的部署有点复杂，相比较单实例数据库的安装而言“事多”一些。机械的劳动就需要两个字——仔细，下面我们就从设计 RAC 的应用环境开始安装 RAC 吧！

1.6.1　设计 RAC 应用环境

在安装 RAC 之前，读者需要对 RAC 的应用环境有个清晰地规划，这样在进一步设计安装组件、任务规划以及安装 RAC 时，就更能清楚地了解每一个步骤应该做什么。总体上，我们设计了两个节点来安装 Clusterware 集群件，并创建一个共享存储用来存储数据库文件，如图 1-7 所示。

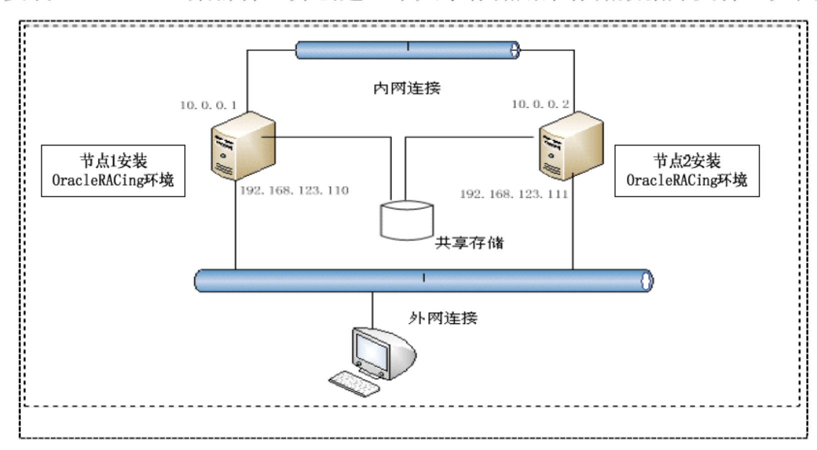


图 1-7　RAC 应用环境示意图

上图中有两台安装 RAC 的主机和两个内部交换机。其中，一台主机配置有公有地址，用于外网；另一台主机配置有一个私有地址，用于内网；一个共享存储用于存储数据库文件。



其实，配置 RAC 需要三个地址：一个 Public 地址、一个 Private 地址、一个 VIP 地址。其中 Public 地址和 VIP 地址在安装 RAC 的过程中会绑定在一起。每个地址都需要一个主机名与之对应。

1.6.2　确认安装的软件组件

为了方便读者学习，作者在虚拟机上模拟出一个安装 RAC 的硬件环境，所以需要一个很重要的虚拟机软件，当然除了虚拟机之外还需要操作系统的支持，本书中我们选择 Linux 系统（AS 4）、Oracle Clusterware、创建自动存储管理的 ASMLib 软件包以及 Oracle 数据库管理软件，软件的具体版本如表 1-1 所示。

表 1-1　部署 RAC 的安装软件列表

软件名称	软件版本
虚拟机软件	VMware-GSX-Server-3.2.0
Linux 系统	RedHat Linux AS 4
Oracle Clusterware	10201_Clusterware_Linux32
ASMLib	oracleasmlib-2.0.2-1.i386 oracleasm-support-2.0.3-1.i386 oracleasm-2.6.9-22.ELsmp-2.0.3-1.i686

（续表）

软件名称	软件版本
Oracle 数据库管理软件	10201_database_linux32
说明：ASMLib 需要与 Linux 系统的版本相对应	

1.6.3 任务规划

我们设计了两个节点 RAC 集群，两个主机分别为一个 RAC 节点，每个节点两个物理网卡。使用 ASM 创建的共享存储设备来存储数据库文件以及参数文件、密码文件，使用 RAW 设备来存储 OCR 和 Voting Disk，下面是具体的规划，分别按 IP 配置、文件分配以及磁盘分区来详细列出。

1．主机的 IP 地址规划表（如表 1-2 和表 1-3 所示）

表 1-2 节点 1 的 IP 配置表

节点 1（myrac1）	对应网卡	IP 地址	网络名
Public IP	Eth0	192.169.123.114	myrac1
Private IP	Eth1	10.0.0.1	myrac1-priv
Visual IP	Eth0	192.168.123.110	myrac1-vip

表 1-3 节点 2 的 IP 配置表

节点 2（myrac2）	对应网卡	IP 地址	网络名
Public IP	Eth0	192.169.123.115	myrac2
Private IP	Eth1	10.0.0.2	myrac2-priv
Visual IP	Eth0	192.168.123.111	myrac2-vip

2．RAC 集群文件分配规划（如表 1-4 所示）

表 1-4 节点 1 的文件分配规划

文件类型	文件分配
Oracle Clusterware	/oracle/product/crs（本地文件）
OCR	/dev/raw/raw1（裸设备）
Voting Disk	/dev/raw/raw2（裸设备）
RDBMS 软件	/oracle/prodct/database（本地文件）
Database files	+DATA（ASM 共享存储）
Backup files	+BACKUP（ASM 共享存储）

我们规划在节点 1 上安装数据库，并使用 ASM 来管理数据库文件，此时在节点 2 上只需要部署 Clusterware 的目录和安装 Oracle 的 RDBMS 软件的目录，这两个目录与节点 1 的文件分配相同。

这里我们就不给出节点 2 的文件分配规格表了。

3．RAC 集群磁盘分区表（如表 1-5 所示）

表 1-5 RAC 集群磁盘分区表

磁盘分区	大小	用途
/dev/sdb1	100MB	存储 Clusterware 的 OCR

（续表）

磁盘分区	大小	用途
/dev/sdb2	100MB	存储 Clusterware 的 Voting Disk
/dev/sdb3	4GB	存储数据库文件（+DATA）
/dev/sdb4	4GB	存储数据库备份文件（+BACKUP）

对磁盘/dev/sdb 进行分区，其中 sdb1 和 sdb2 分别存储 OCR 和 Voting Disk 的内容，而 sdb3 和 sdb4 用来存放数据库的各种文件（数据文件、控制文件、参数文件等）和备份文件。

1.6.4　安装虚拟机

考虑到很多读者没有安装 RAC 的一个物理环境，或者虽然具有这样的环境但是没有机会从头到尾安装配置一次 RAC 环境，所以，笔者提供了一个虚拟机环境，通过创建两个虚拟机，并在虚拟机上安装 Linux 系统、在系统上安装 Clusterware 以及数据库软件，就可以轻松创建一个 RAC 环境。我们使用虚拟机 VMware GSX Server，版本为 3.2.0。

安装过程如下所示。

01 打开安装好的 VMware GSX Server 虚拟机软件，如图 1-8 所示。

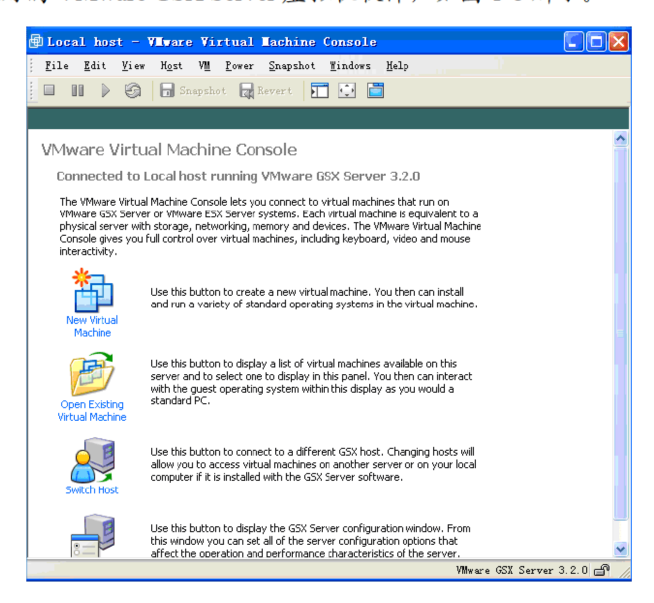


图 1-8　VMware GSX Server 虚拟机控制台

02 单击“New Virtual Machine”图标创建新的虚拟机，弹出如图 1-9 所示的对话框。

03 连续单击“下一步”按钮分别显示如图 1-10 和图 1-11 所示的窗体，在图 1-10 中根据需求选择适合的类型：Typical 或者 Custom，笔者选择 Custom。图 1-11 中要求选择客户操作系统，即在虚拟机上要安装的操作系统，我们选中“Linux”单选按钮，并在下拉列表中选中“Red Hat Enterprise Linux 4”。

04 继续单击“下一步”按钮，打开如图 1-12 和图 1-13 所示的窗体，在图 1-13 中设置虚拟机的名字以及虚拟机安装目录，图 1-14 中设置启动或关闭方式，笔者的选择如图 1-12、图 1-13 所示。



图 1-9　启动 GSX Server 3 创建虚拟机

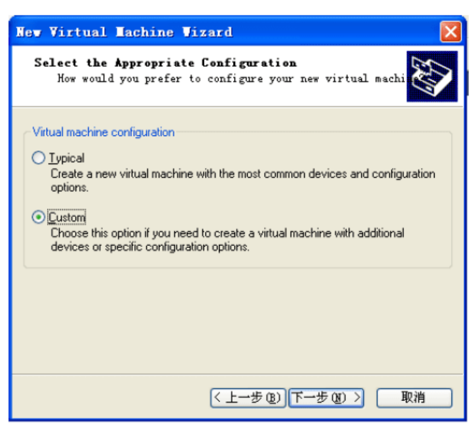


图 1-10　选择“Custom”单选按钮

图 1-11　选择客户操作系统



图 1-12　命名虚拟机

图 1-13　设置虚拟机启动、关闭选项

05 继续单击“下一步”按钮，打开如图 1-14 所示的窗体，设置虚拟机内存，这里最少设置 1G 的大小。如果今后扩展了内存，也可以通过修改虚拟机设置来修改内存大小。

06 继续单击“下一步”按钮，分别依序打开如图 1-15~1-19 所示的窗体。图 1-15 中设置网络类型，我们选中“Use Bridged networking”单选按钮。图 1-16 中选择 I/O 适配器的类型，选中“LSI Logic”单选按钮。图 1-17 中选择磁盘方式，有三个选项，选中“Create a new virtual disk”单选按钮。图 1-18 中选择虚拟机类型，我们选择 SCSI 类型。而图 1-19 中设置磁盘大小，设置大小为 10GB，并选中“Allocate all disk space now.”复选框。

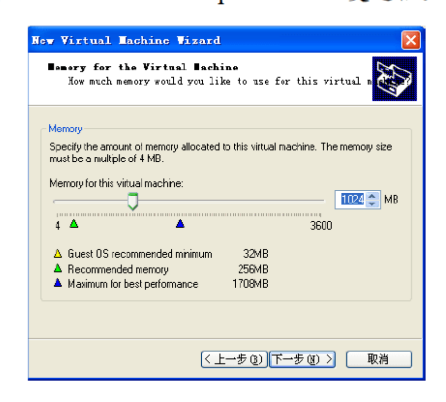


图 1-14 设置虚拟机内存

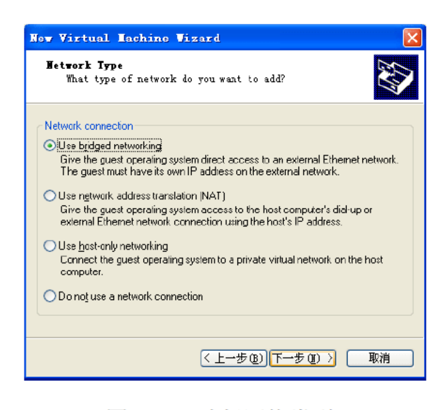


图 1-15 选择网络类型

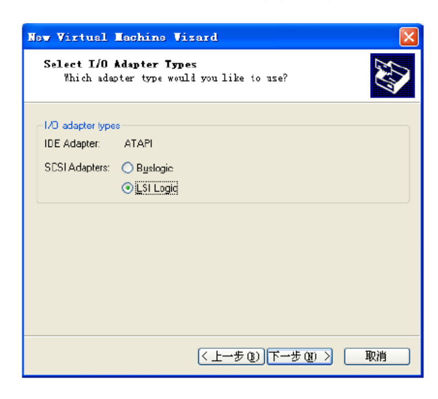


图 1-16 选择 I/O 适配器类型

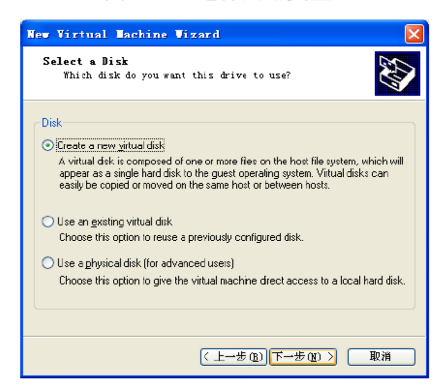


图 1-17 创建虚拟磁盘

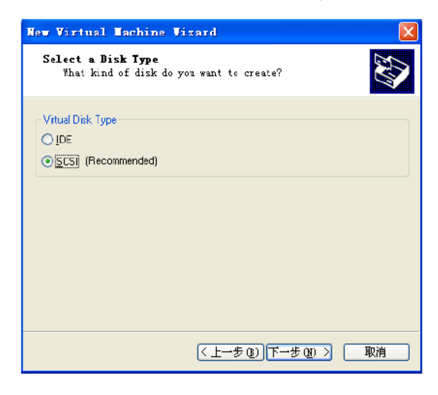


图 1-18 选择虚拟磁盘类型



图 1-19 设置虚拟磁盘大小

07 单击如图 1-19 所示的“下一步”按钮，打开如图 1-20 所示的窗体，设置磁盘文件名，即

该磁盘在系统的文件表示，然后单击“完成”按钮，打开如图 1-21 所示的窗体，开始创建磁盘，这个创建过程会根据磁盘大小时间长短不同。

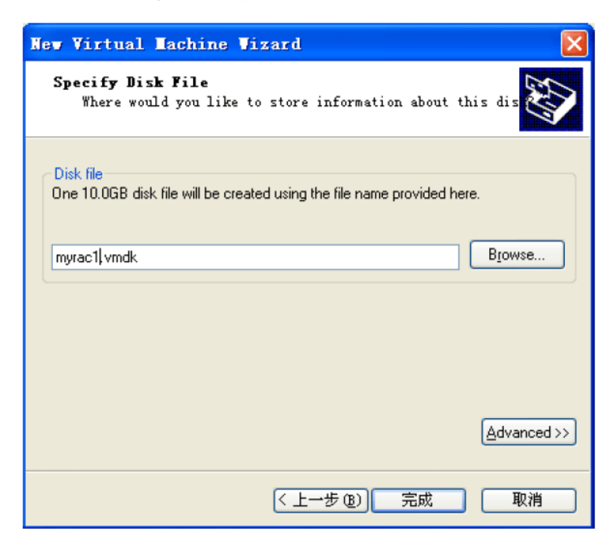


图 1-20 指定虚拟磁盘文件名

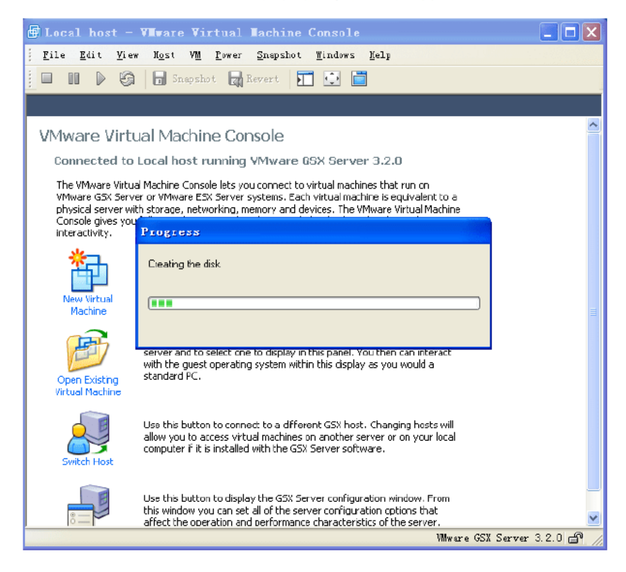


图 1-21 开始创建磁盘

08 在磁盘创建完后，自动进入如图 1-22 所示的对话框。从“Devices”选项组中可以清晰地看出该虚拟机的硬件配置。

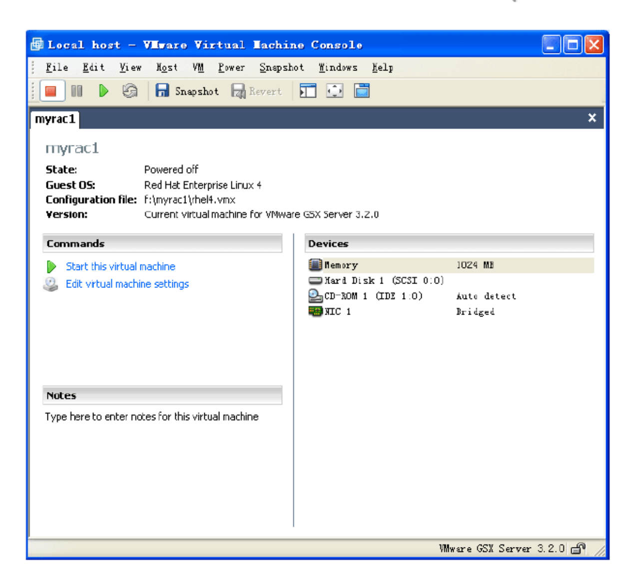


图 1-22　安装好的虚拟机

1.6.5　在虚拟机上安装 Linux 操作系统

如果读者有 RedHat AS 4 的.iso 格式的系统文件，则可以直接将该文件加载到如图 1-23 所示的“Use ISO image”单选按钮下。如果下载的是分割开的多个.iso 文件，则需要将多个.iso 文件制作为一个 iso 文件，或直接使用一张系统光盘。

下面演示如何在虚拟机上安装 Linux 操作系统，这里需要读者认真学习安装过程，尤其是在分区选择时，如何分区以及分区大小都是有要求的。

具体安装步骤如图 1-23~图 1-27 所示。

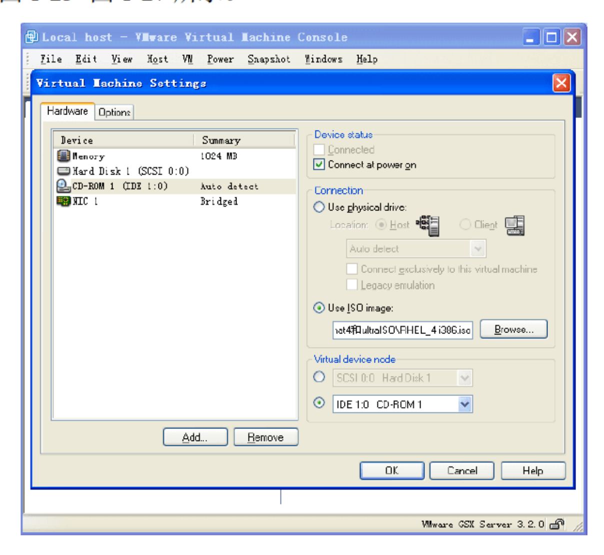


图 1-23　加载操作系统的 ISO 映像文件

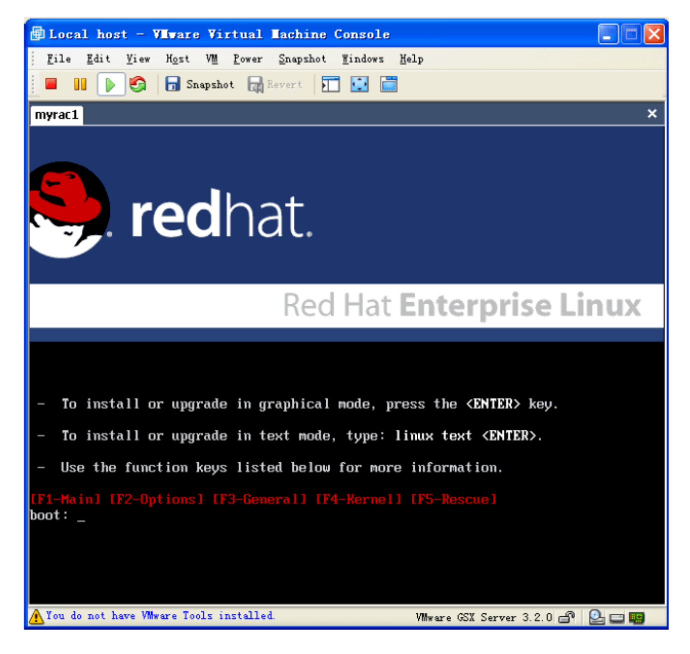


图 1-24 进入引导系统启动

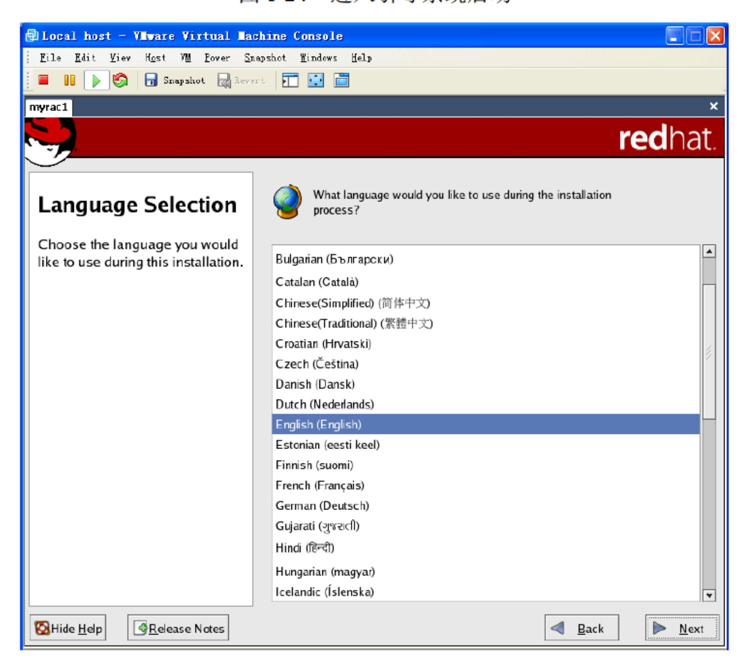


图 1-25 选择安装过程中使用的语言

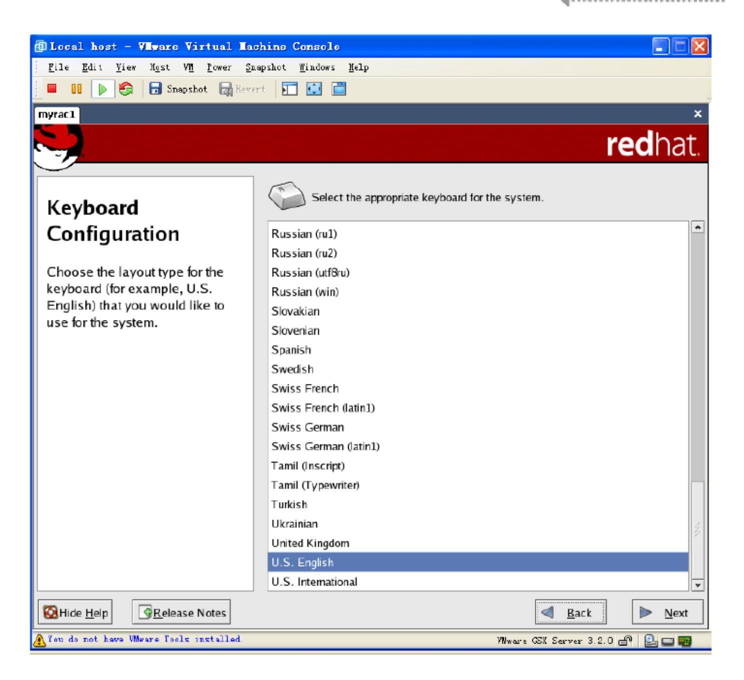


图 1-26　选择合适的键盘类型

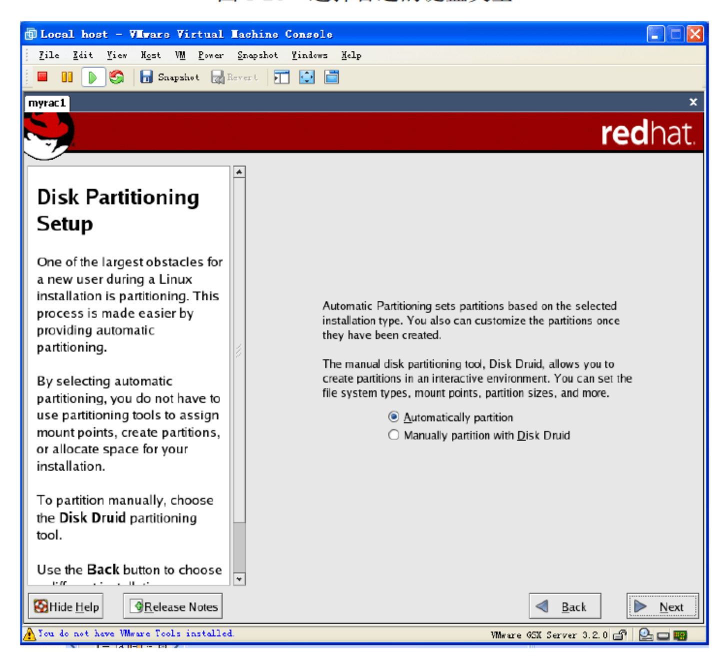


图 1-27　选择分区方式

在分区方式中，要选择手工分区，即“Manually partition with Disk Druid”单选按钮，然后进入分区主界面，如图 1-28 所示。

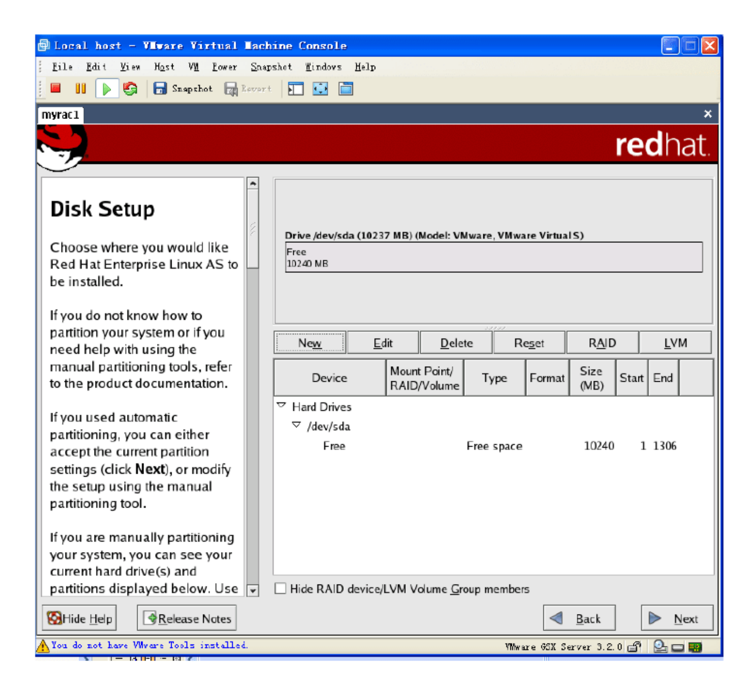


图 1-28　进入分区主界面

单击“New”按钮，弹出如图 1-29 所示的界面。

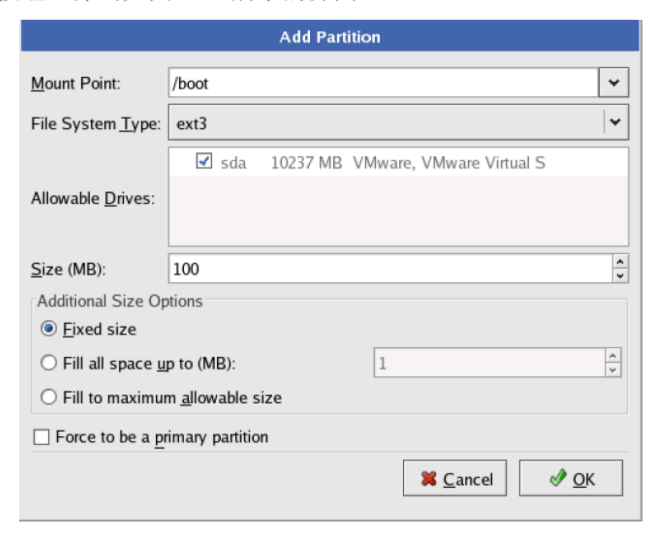


图 1-29　设置“/boot”分区

“/boot”分区是系统启动分区，不需要很大的空间，这里设该分区大小为 100MB。还需要设置根目录“/”、“/oracle”目录以及 swap 分区。参考图 1-29 依次设置这些分区目录，结果如图 1-30 所示。



图 1-30　设置好的分区结果

单击“Next”按钮进入如图 1-31 所示的界面。

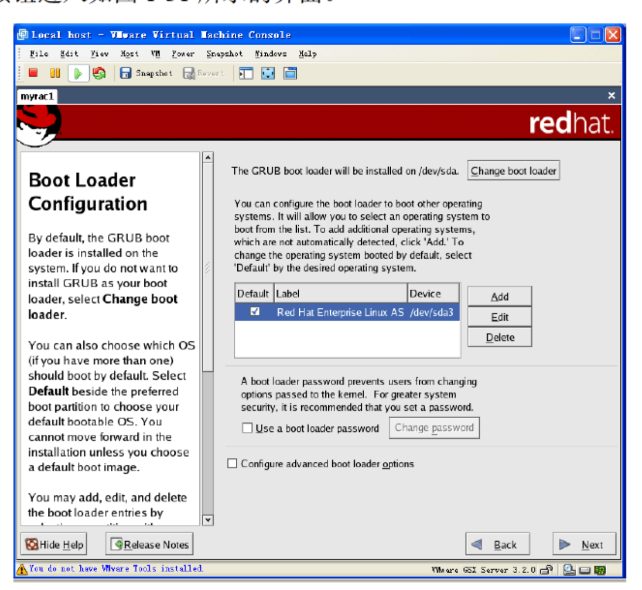


图 1-31　选择安装系统的分区

系统默认将操作系统安装在设备“/dev/sda3”上，而挂接点为“/”，在图 1-31 中可以清楚看出这一设置结果。接下来单击如图 1-31 所示的“Next”按钮，打开如图 1-32 所示的窗体。单击“Edit”按钮配置 IP 和主机名，并选中“manually”单选按钮设置主机名，并设置网关。这里一旦配置了 IP 就自动增加了一块网卡。

继续单击“Next”按钮，进入如图 1-33 所示的界面，在界面上选中“No firewall”单选按钮，

并在下方的“Enable Selinux?”下拉列表中选中“Disable”。

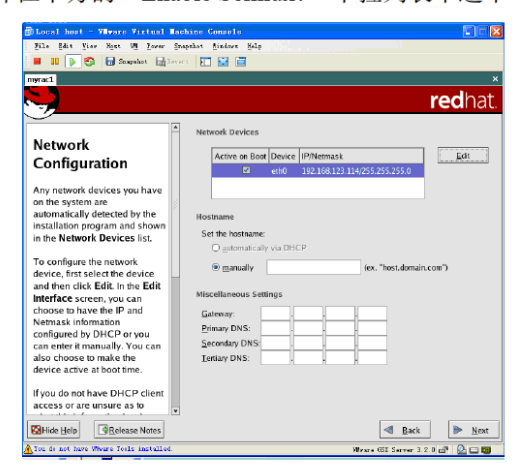


图 1-32 配置 IP、主机名

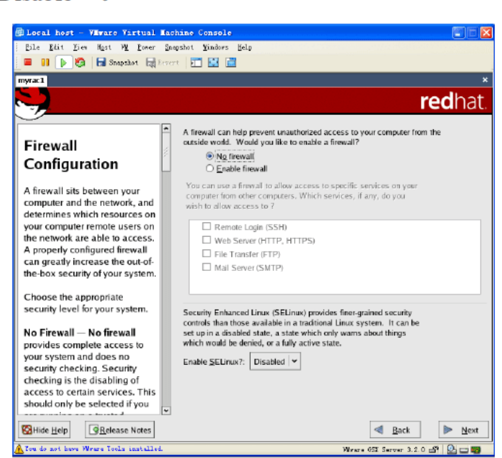


图 1-33 关闭 Linux 的防火墙

继续单击“Next”按钮，进入设置语言界面，选择系统的默认语言，如图 1-34 所示，此时选中“English（USA）”复选框。

继续单击“Next”按钮，进入如图 1-35 所示的界面，设置 root 用户的用户名和密码，该用户拥有对操作系统的所有权限。

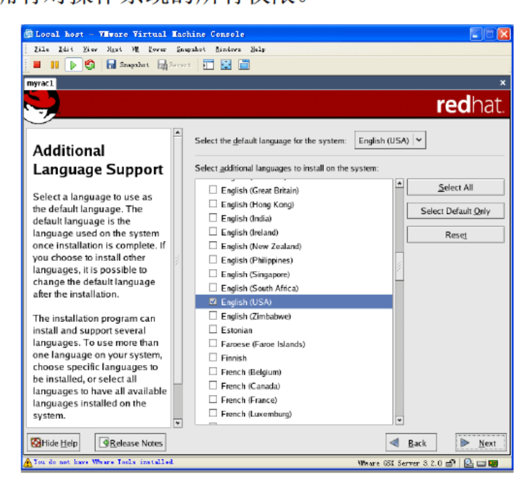


图 1-34 设置操作系统的语言



图 1-35 设置 root 用户的密码

在设置好 root 用户的密码后就可以安装操作系统了，如图 1-36 所示。

在安装完成后需要重新启动（reboot）系统，而后按照提示做简单的配置，即可完成系统的安装，如图 1-37 所示。

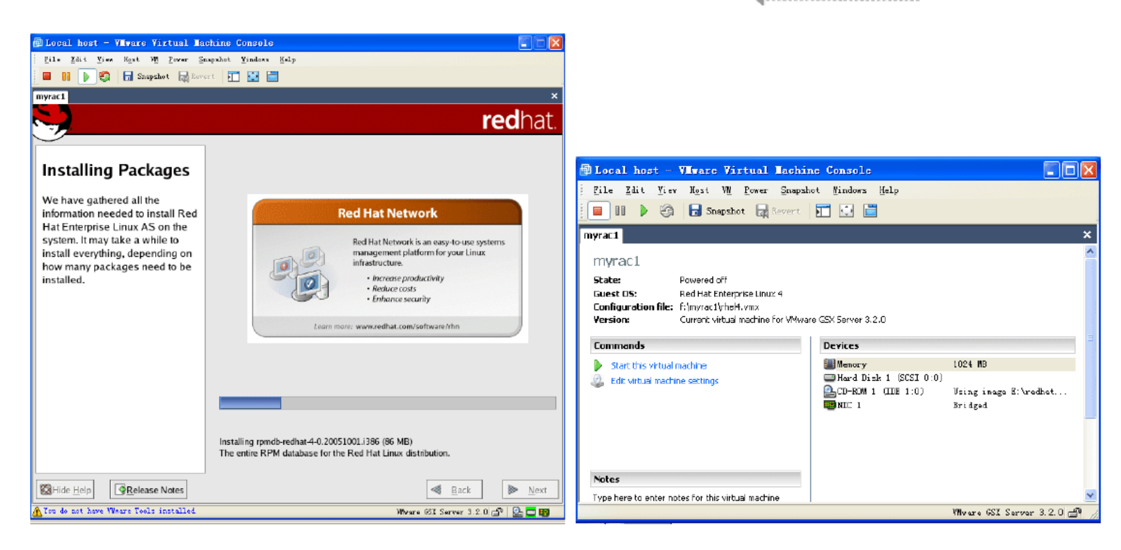


图 1-36　系统的安装过程　　　　图 1-37

在图 1-32 中，通过配置 IP 已经自动创建了一块网卡，下面我们添加第二块网卡，过程与在虚拟机上添加硬件的步骤相同。我们单击图 1-37 中的“Edit virtual machine settings”选项，弹出如图 1-38 所示的对话框。

单击“Add”按钮来增加相应的系统组件，如硬盘、网卡等，如图 1-39 所示。

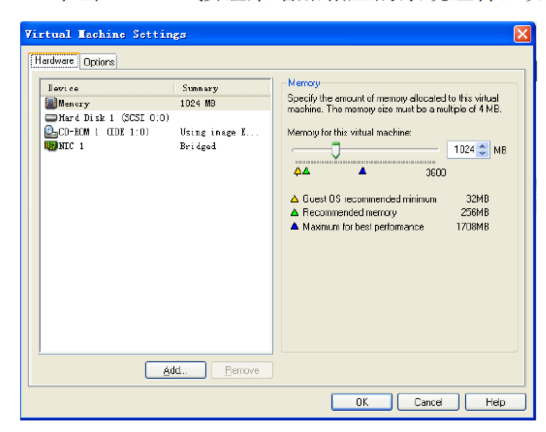


图 1-38　添加虚拟机硬件

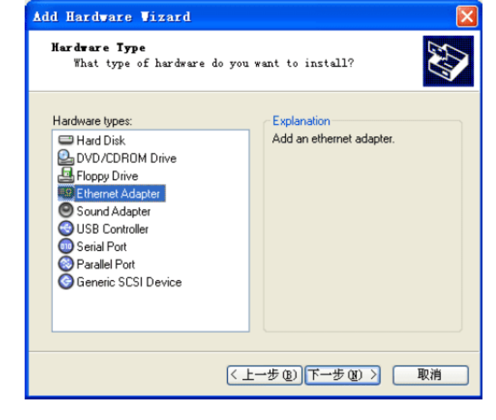


图 1-39　选择添加的硬件

选中“Ethernet Adapter”选项，然后单击“下一步”按钮，弹出如图 1-40 所示对话框。

选中“Bridged:Connected directly to the physical network”单选按钮，即系统加电时自动连接网络，单击“完成”按钮，添加结果如图 1-41 所示。

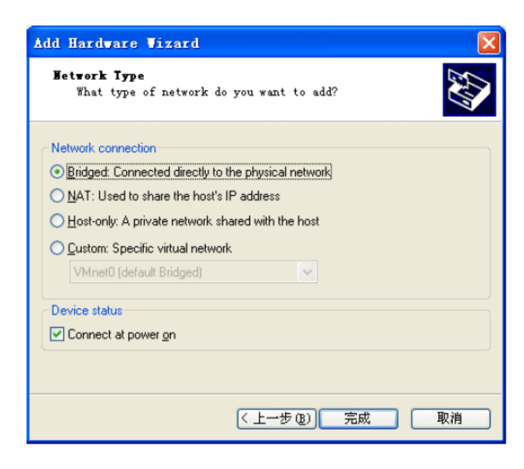


图 1-40　设置网络连接方式

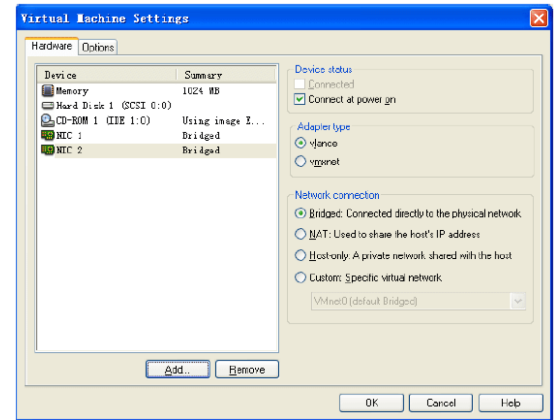


图 1-41　添加了一块新网卡

接下来我们设置共享存储时两个节点都可以访问的存储区，即用来存储 Clusterware 的 OCR 和 Voting Disk。与添加网卡类似，我们添加一块磁盘，操作过程如图 1-42~图 1-47 所示。

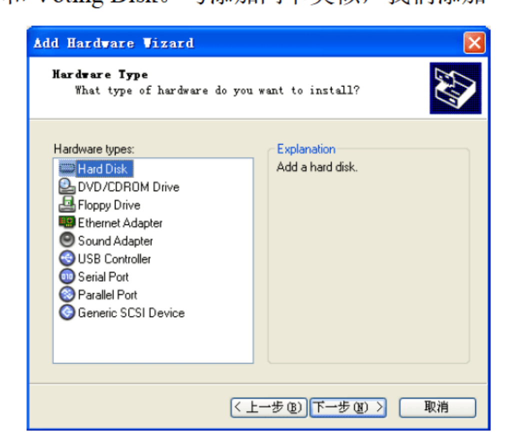


图 1-42　添加共享磁盘

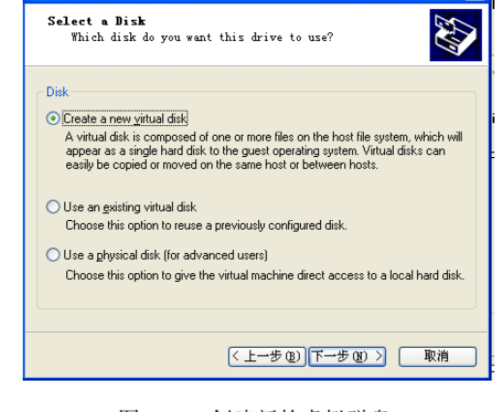


图 1-43　创建新的虚拟磁盘

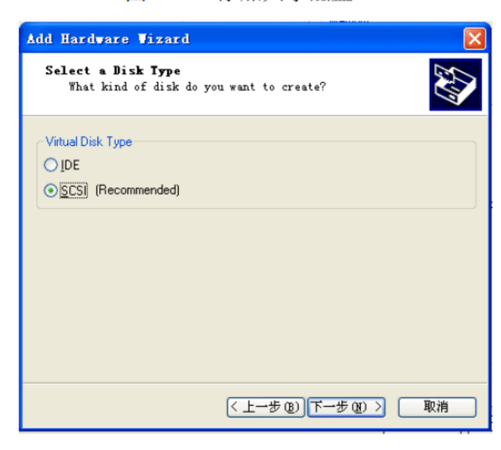


图 1-44　选择虚拟磁盘类型

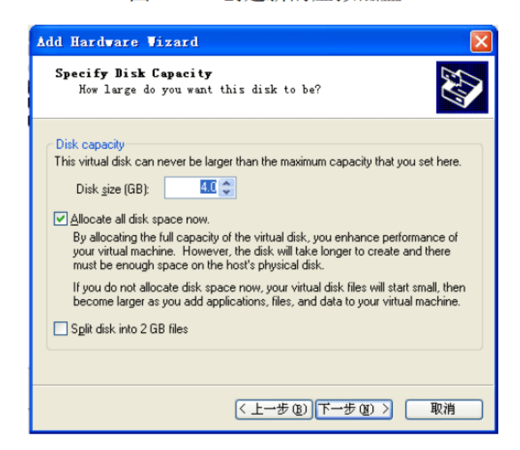


图 1-45　设置虚拟磁盘容量

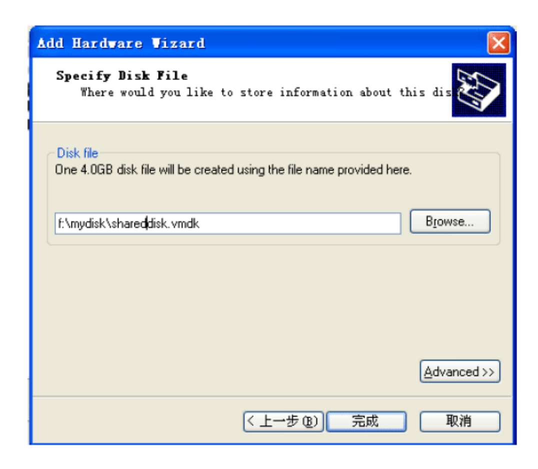


图 1-46　设置虚拟磁盘文件名

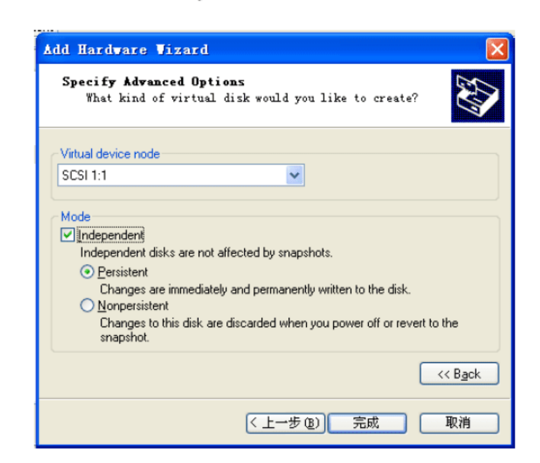


图 1-47　设置虚拟磁盘的高级选项

单击“完成”按钮则进入创建过程，如图 1-48 所示。

如图 1-48 所示的创建磁盘过程结束后，自动进入如图 1-49 所示的对话框，提示当前虚拟机的完整硬件配置。

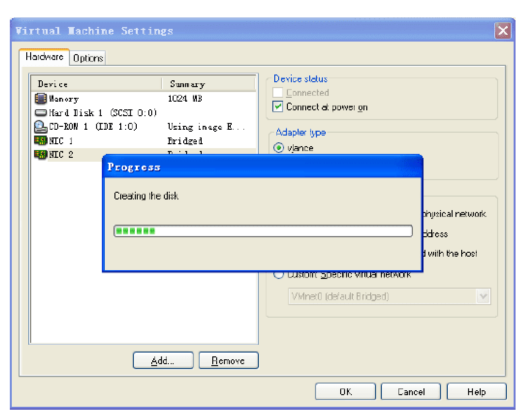


图 1-48　创建共享磁盘的过程

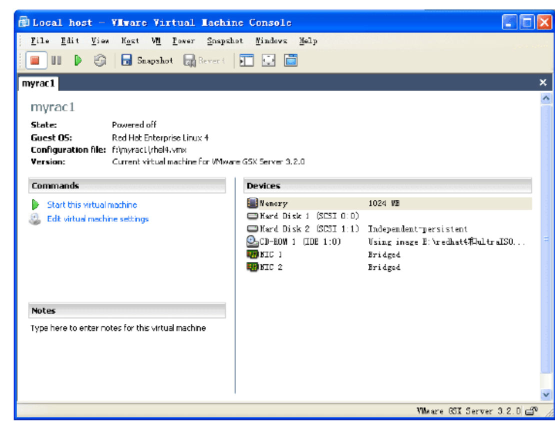


图 1-49　当前虚拟机的完整配置

在创建完成一个虚拟机后，我们通过拷贝虚拟机文件的方式来创建另一个虚拟机，不过需要提前进行一些设置。

首先需要对磁盘文件做修改，在笔者的计算机上需要修改的文件如图 1-50 所示。

使用记事本打开文件 rhel4.vmx，添加记录“disk.locking="FALSE"”，并在 scsi1 磁盘信息区添加一行记录“scsi1.sharedBus="virtual"”，其目的是使得该磁盘在两个虚拟机之间得以共享。

然后将目录 F:\myrac1 下的所有文件拷贝到 F:\myrac2 目录下，之后设置文件 rhel4.vmx 中的参数 displayName，修改结果为“displayName="myrac2"”。其目的是设置虚拟机的名字。之后打开已经存在的虚拟机，如图 1-51 所示。

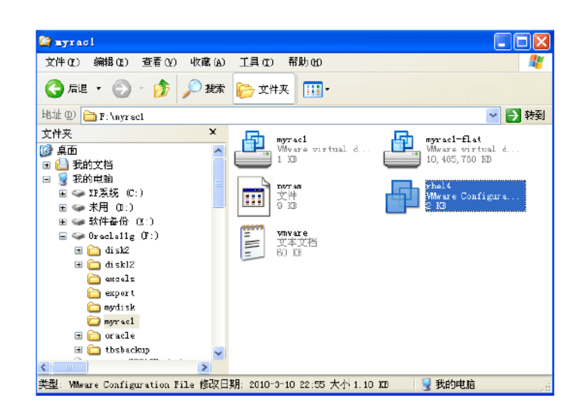


图 1-50　完整的虚拟机文件

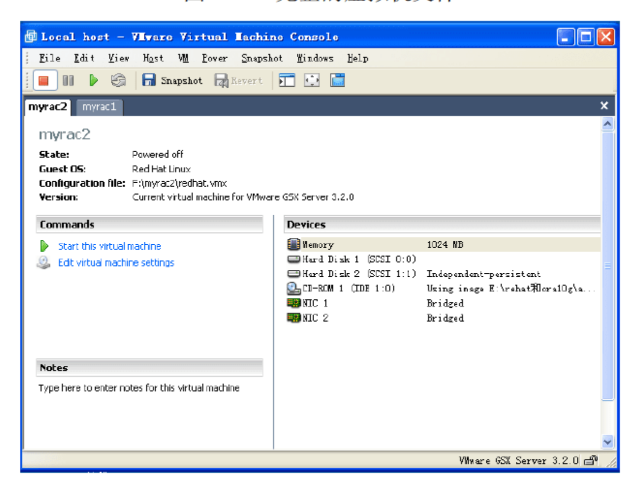


图 1-51　两个虚拟机的完整配置

1.6.6　配置主机

主机配置是一件很繁琐的过程，读者需要细心对待，下面分步骤实现主机配置。

1．主机需要设置三个网络地址

三个网络地址包括：Public 地址、Private 地址和 VIP 地址，其中 Public 地址和 VIP 地址分配到同一个 Public 网卡上。我们在主机上添加两个网卡 eth0 和 eth1，在 eth0 网卡绑定 Public 地址和 VIP 地址，eth1 网卡设置 Private 地址。

所以，首先需要保证在两个主机上有两块物理网卡。在创建虚拟机时，已经创建了两个网卡，现在的任务是按照规划设置网卡地址，这里只给出一个例子，就不一一设置任务计划中的网卡设置了。

01 在虚拟机主界面选择 Applications→System Tools→Network Device Control，弹出如图 1-52 所示的对话框。

02 单击图 1-52 右边的“Configure…”按钮。此时，在虚拟机 myrac1 上由于已经安装了两

个网卡，同时在创建虚拟机时已经配置了一块网卡 eth0，所以只需要配置另一个网卡 eth1。单击图 1-53 中的“New”按钮来配置新的网卡，界面如图 1-54 所示。

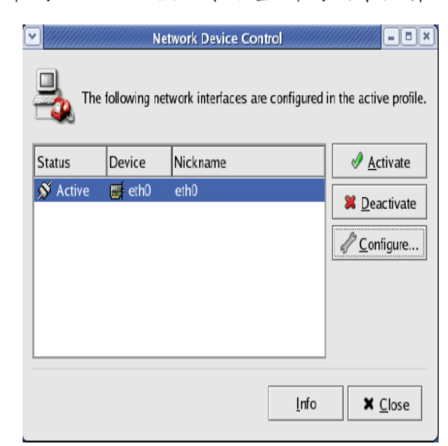


图 1-52

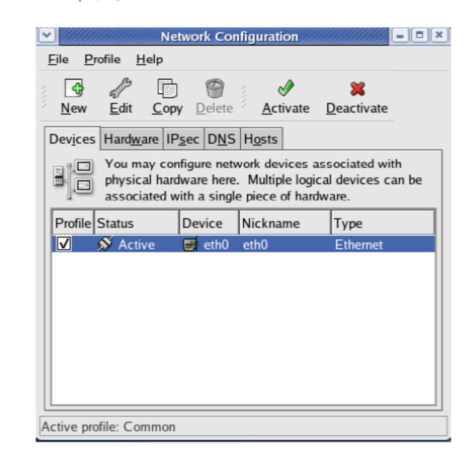


图 1-53　网络配置

03　然后单击“Forward”按钮，弹出如图 1-55 所示对话框。

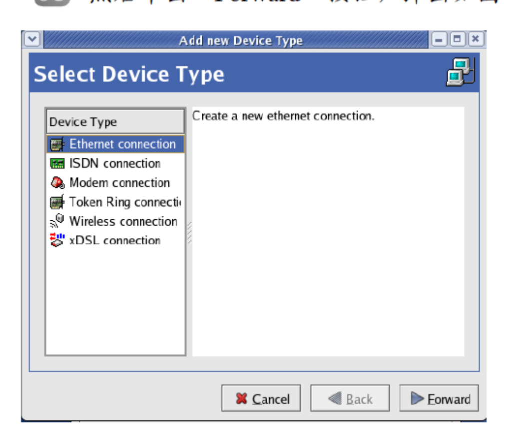


图 1-54　选择设备类型为 Ethernet connection

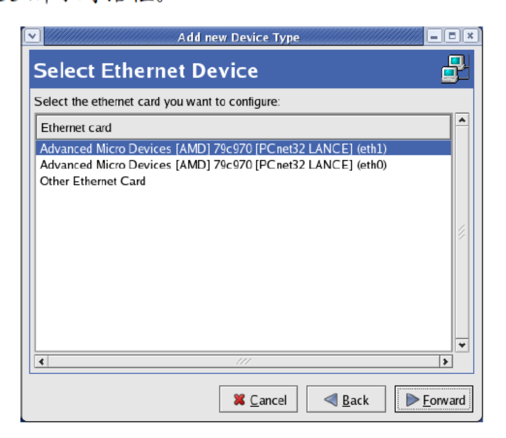


图 1-55　选择需要配置的网卡

04　单击“Forward”按钮，弹出如图 1-56 所示的对话框。

05　单击“Forward”按钮，会弹出一个地址配置清单，然后应用该设置，最后回到如图 1-57 所示的对话框。



此时的网卡处于“Inactive”状态，可以通过单击“Activate”按钮来激活该网卡，此时完成网卡地址的配置，用同样的方法在节点 myrac2 上配置网卡 eth1，并修改网卡 eth0 的 IP 地址配置（因为其 IP 地址和虚拟机 myrac1 上的相同）。

06　在设置好网卡后，可以测试配置是否正确以及是否启动网卡，测试方法如图 1-58 所示。

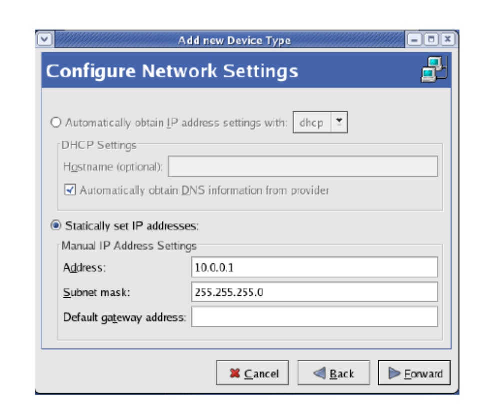


图 1-56　设置网卡 eth1 的地址

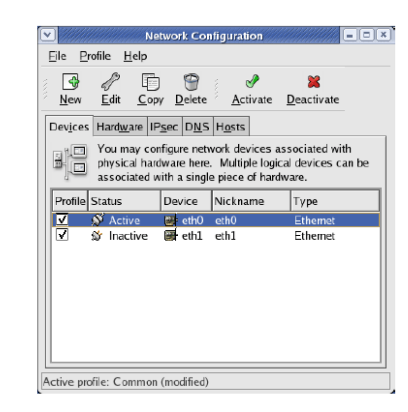


图 1-57　完成网卡配置

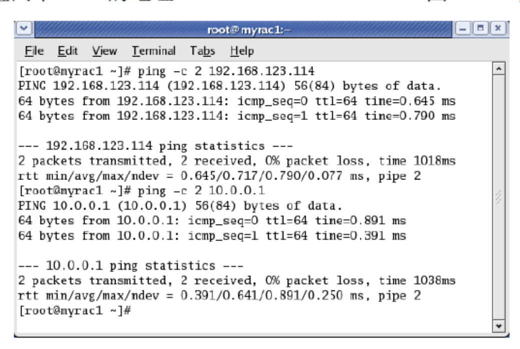


图 1-58　测试网络连通性

2．编辑 hosts 文件设置

需要在 hosts 文件中设置网卡的地址和对应的网络名称，这些配置在两个节点是相同的，其中 VIP 是 Clusterware 安装过程中创建的。在节点 myrac1、myrac2 中编辑文件/etc/hosts。

该操作需要在 root 用户下进行，指令如下所示。

```
[root@myrac1 ~]# vi  /etc/hosts
```

hosts 文件修改后的配置如图 1-59 所示。修改后同样使用“ping –c 2 myrac1”指令测试连通性。

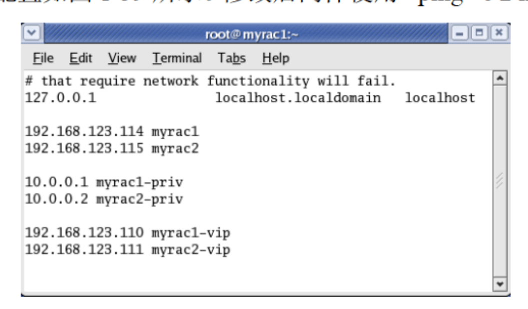


图 1-59　修改后的 hosts 文件

3．安装系统所需要的 RPM 包

该步骤需要在两个节点上执行。此时需要根据操作系统的版本来安装不同的 RPM 包，笔者的

Linux 核心版本为 2.6.9-22，所需要的安装包如下所示。

```
compat-libstdc++-33-3.2.3-47.3.i386.rpm
gcc-3.4.4-2.i386.rpm
gcc-c++-3.4.4-2.i386.rpm
glibc-devel-2.3.4-2.13.i386.rpm
glibc-headers-2.3.4-2.13.i386.rpm
glibc-kernheaders-2.4-9.1.9.8.EL.i386.rpm
libstdc++-devel-3.4.4-2.i386.rpm
sysstat-5.0.5-1.i386.rpm
```

接着就可以使用 rpm 指令来安装这些 RPM 包，这些包在 Linux 的安装盘里可以找到。笔者的系统是通过加载.ISO 系统映像来安装的，所以进入映像文件就可以找到这些 RPM 包，如图 1-60 所示。

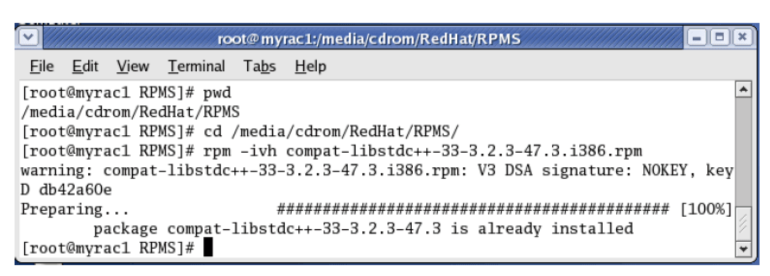


图 1-60　安装*.rpm 包

在上图中首先进入映像文件所在的安装包的目录，即使用“cd /media/cdrom/RedHat/RPMS/”指令，然后使用“rpm –ivh *.rpm”指令来依次安装需要的安装包。

4. 创建 oracle 用户、创建软件安装目录

（1）创建 oracle 用户以及用户组

本步骤使用 groupadd 以及 useradd 分别创建 oracle 用户组以及 oracle 用户，要求安装 RAC 的每一个节点上 UID（用户 ID）和 GID（组 ID）都相同。以下指令需要在每个节点上执行。

首先创建用户组，如下例所示。

例子 1-1 创建 DBA 用户组。

```
[root@myrac1 ~]groupadd -g 700 dba
```

其次创建 oracle 用户，如下例所示。

例子 1-2 向用户组增加新用户。

```
[root@myrac1 ~]useradd -u 500 -g dba oracle
```

此时成功创建了用户 oracle，处于安全的目的，我们需要继续设置该用户的密码，如下例所示。

例子 1-3 设置用户密码。

```
[root@myrac1 ~]passwd oracle
Changing password for user oracle.
```

```
New UNIX password:
Retype new UNIX password:
Passwd: all authentication tokens updated successfully.
```



在输入用户密码时，没有显示，如果两次输入不一致，则有错误提示，需要重新设置。

（2）创建文件目录，此处需要创建两个目录以及修改一个目录的归属

在每个节点上执行如下相同的操作。创建两个目录，一个为 Clusterware 的安装目录“/oracle/product/crs”，一个为数据库的安装目录“/oracle/product/database”，如下例所示。

例子 1-4 创建目录并设置目录权限。

```
[root@myrac1 ~]mkdir -p /oracle/product/crs
[root@myrac1 ~]mkdir -p /oracle/product/database
[root@myrac1 ~]chown -R oracle:dba /oracle/product/crs
[root@myrac1 ~]chown -R oracle:dba /oracle/product/database
[root@myrac1 ~]chmod 775 /oracle/product/crs
[root@myrac1 ~]chmod 775 /oracle/product/database
```

修改目录“/oracle”的用户归属，如下例所示。

```
[root@myrac1 ~]chown  -R  oracle:dba  /oracle
[root@myrac1 ~]chmod  775  /oracle
```

在设置完后，我们应该使用“ls –l”指令查看修改结果，如图 1-61 所示。

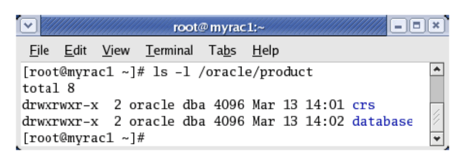


图 1-61　查看目录权限

5. 配置环境变量以及设置用户资源

读者应该根据不同的 shell 类型编辑 profile 文件，其作用是配置环境变量，如 Clusterware 的安装路径、数据库的安装路径以及定义 PATH 环境变量，如图 1-62 所示。

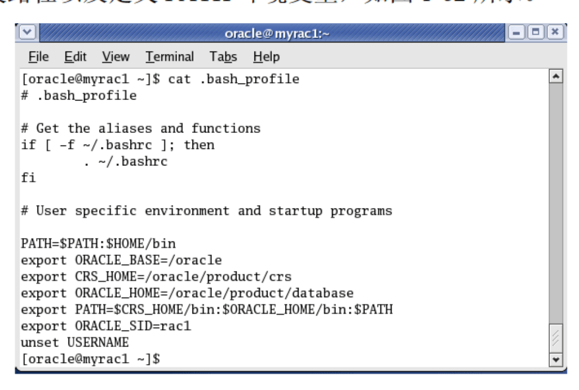


图 1-62　编辑.bash_profile 文件

接着设置用户资源，其目的是限制 oracle 用户对系统资源的使用，需要编辑文件 limits.conf，如图 1-63 所示。

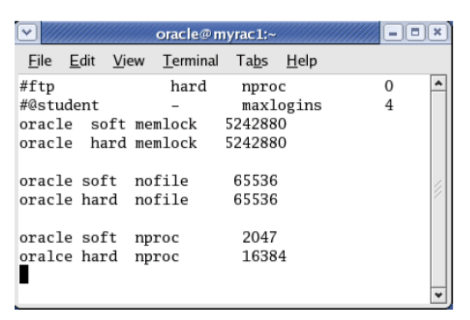


图 1-63　编辑 limits.conf 文件

6. 配置 Linux 系统内核参数

修改内核参数就是修改/etc/sysctl.conf 文件，在文件中添加如下内容。

```
kernel.shmmax=2147483648
kernel.sem=250 32000 100 128
fs.file-max=65536
net.ipv4.ip local port range=1024 65000
net.core.rmem default=262144
net.core.wmem default=262144
net.core.rmem max=262144
net.core.wmem_max=262144
```

最后在 root 用户下执行如下指令，使得内核参数修改生效。

```
sysctl -p
```

7. 配置 SSH 用户信任关系

配置 SSH 的目的就是使得两个节点主机之间的相同用户可以无障碍的通信，在安装 Clusterware、安装数据库软件，以及配置监听等操作时，都需要两个节点上的用户之间可以无障碍通信。下面是配置 SSH 的要求及方法。

（1）配置 SSH 用户等价性前提条件。

- 指定建立 SSH 等价性的用户必须具有相同的用户名、用户 ID 以及相同的密码。
- 指定建立 SSH 等价性的用户必须属于相同的用户组。
- 指定建立 SSH 等价性的用户必须具有相同的用户组 ID。

（2）RSA 和 DSA 的作用。

为了配置 SSH，必须首先在两个集群环境下的节点上创建 Rivest-Shamir-Adleman (RSA)密钥，和 Digital Signature Algorithm (DSA)密钥，一旦创建了公有密钥和私有密钥，即可将所有节点上的密钥拷贝到一个授权文件（authorized files），再将这个文件拷贝回所有节点的原密钥所在目录。

（3）重新使用 oracle 用户登录，在配置 SSH 前必须先退出 root 用户会话，再用 oracle 用户登录，建立新的 oracle 用户的会话环境。

SSH 用户等价性的目的是建立不同节点的两个用户之间的信任关系，一个节点通过网络访问另一个节点时就不需要密码认证，而是直接可以访问对方的文件，如在安装 Clusterware 时，在一个节点安装的文件会默认拷贝到 RAC 环境中的另一个节点的相同目录下，而此时是不允许用户通过某种方式输入用户口令的，通过建立 SSH 信任关系就可以直接将文件拷贝的另一个主机相同用户的相同目录下。

（4）配置 SSH。

我们先在主机 myrac2 中的 oracle 用户下建立 SSH 用户等价性的前期工作，相关指令如下所示。

```
[oracle@myrac2~]$ cd
[oracle@myrac2~]$ mkdir .ssh
[oracle@myrac2~]$chmod 700 .ssh
[oracle@myrac2~]$cd .ssh
[oracle@myrac2.ssh]$ ssh-keygen -t  rsa
[oracle@myrac2.ssh]$ ssh-keygen -t  dsa
[oracle@myrac2.ssh]$ cat *.pub > authorized_keys
```

同样在节点 myrac1 上做相同的工作，然后将节点 myrac2 中的 authorized_keys 文件拷贝到节点 myrac1 中的/home/oracle/.ssh/目录下，文件名改为 keys_myrac2。目的是在节点 myrac1 将两个节点的 authorized_keys 的文件内容合并成一个文件，如图 1-64 所示，将 myrac2 节点的 authorized_keys 文件拷贝到主机 myrac1 的/home/oracle/.ssh/目录下。

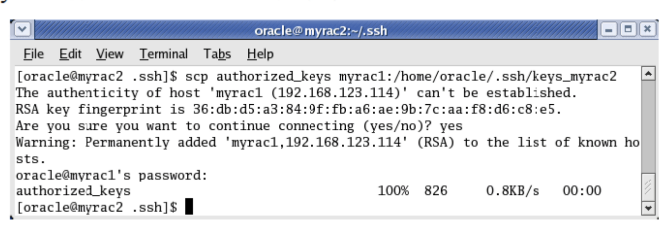


图 1-64　拷贝 authorized_keys 文件

然后在节点 myrac1 做一次文件合并的工作，操作方式如下例所示。

```
[oracle@myrac1.ssh]$ cat keys_myrac2 >> authorized_keys
```

再将该文件从节点 myrac1 拷贝到节点 myrac2，这样该文件就覆盖了节点 myrac2 上原有的 authorized_keys 文件，操作方式如图 1-65 所示。

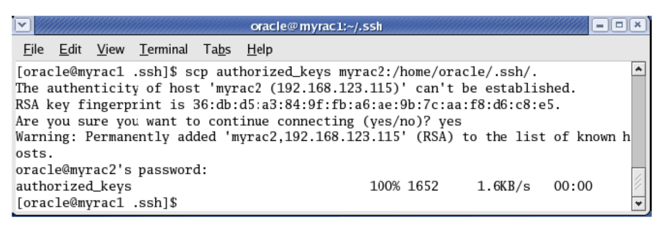


图 1-65　向 myrac2 节点拷贝文件

此时，我们就完成了 SSH 用户等价性的配置。下面我们需要验证这个等价关系。首先在节点 myrac1 执行如下操作：

```
[oracle@myrac1]$ ssh myrac2
```

上述指令的执行过程如图 1-66 所示。

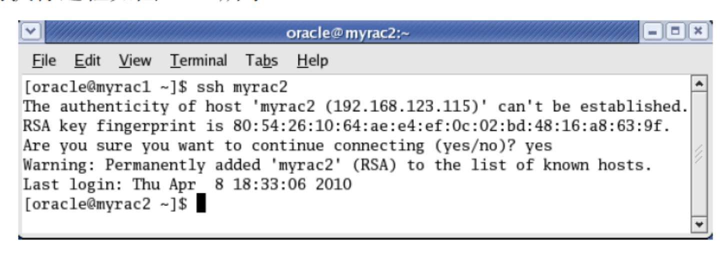


图 1-66　建立到节点 myrac1 的 ssh 信任关系

接下来执行如下指令：

```
[oracle@myrac1~]$ ssh myrac1-priv
```

上述指令建立到节点 myrac1 的信任关系，使用私有地址，如图 1-67 所示。

```
oracle@myrac1:~
File Edit View Terminal Tabs Help
[oracle@myrac1 ~]$ ssh myrac1-priv
The authenticity of host 'myrac1-priv (10.0.0.1)' can't be established.
RSA key fingerprint is 80:54:26:10:64:ae:e4:ef:0c:02:bd:48:16:a8:63:9f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'myrac1-priv,10.0.0.1' (RSA) to the list of kn
own hosts.
Last login: Thu Apr  8 18:04:10 2010
[oracle@myrac1 ~]$
```

图 1-67　建立到节点 myrac1 的 ssh 信任关系

其次，在节点 myrac1 上执行如下操作：

```
[oracle@myrac2]$ ssh myrac1
[oracle@myrac2]$ ssh myrac1-priv
```



如果 SSH 用户的等价性配置正确，上述操作均可以成功登录对方主机，上述操作均是在 oracle 用户下的操作。如果读者忘记该操作在 root 用户下做了等价性配置，显然在使用 oracle 用户安装 Clusterware 时，无法向对方拷贝安装文件。

8. 配置节点间时钟同步

在安装 RAC 时，如果节点之间不能实现时钟的同步，则无法成功安装 Clusterware，会提示错误。

而设置节点间同步有很多方法，如果读者的计算机和 Internet 相连，则可以使用一个标准时钟服务器，也可以在系统运行环境内指定一个时钟服务器。但是，在我们当前学习的环境中，还是推荐使用两个节点中的一个作为时钟服务器，而另一个节点主机向它同步。这里，我把 myrac1 作为时钟服务器，而 myrac2 向该时钟服务器同步。下面演示设置过程。

首先，在节点 myrac1 上使用 vi 编辑器来编辑文件**/etc/ntp.conf**，文件修改后的结果如下所示。

```
server 127.127.1.0
dudge 127.127.1.0 stratum 11
broadcastdelay 0.008
```

其次，在节点 myrac2 上使用 vi 编辑器来编辑文件/etc/ntp.conf，文件修改后的结果如下所示：

```
server 192.168.123.114 prefer
driftfile  /var/lib/ntp/drift
broadcastdelay 0.008
```



上述在节点 myrac1 和节点 myrac2 中执行的指令都是在 root 用户下，其中 myrac2 节点中 prefer 的意思是该节点首先选择该服务器作为时钟服务器。

在修改完成后重新启动 ntp 服务，指令如下所示。

```
[root@myrac2 ~]# service ntpd restart
```

在完成上述时钟同步设置后，可以通过如下方式验证设置结果。

```
[oracle@myrac1 ~]$ ssh myrac2 date
dTue Mar 16 19:56:19 CST 2010
[oracle@myrac1 ~]$ date
dTue Mar 16 19:56:18 CST 2010
```

从上面输出可见，节点 myrac1 和节点 myrac2 实现了时间同步。

9. 配置 hangcheck-timer。

hangcheck-timer 模块负责监控 Linux 系统的内核状态。它需要两个参数来设置检查周期，一个是 hangcheck-tick，它的作用是告诉 hangcheck-timer 模块多长时间检查一次内核健康，一个是 hangcheck_margin，它定义区间，就是如果内核检查没有成功则在这个时间范围内再次检查成功则说明内核运行正常，否则会认为 Linux 系统内核运行异常，会自动重启 Linux 系统。显然在一个 24×7 小时运行的商业数据库系统上，系统重启或许是灾难性的。下面是设置这两个参数的步骤。

01 在文件/etc/rc.d/rc.local 中添加参数，使得系统在启动时加载 hangcheck-timer 模块。使用 vi 编辑器编辑文件/etc/rc.d/rc.local，增加如下指令，操作界面如图 1-68 所示。

```
modprobe hangcheck-timer
```

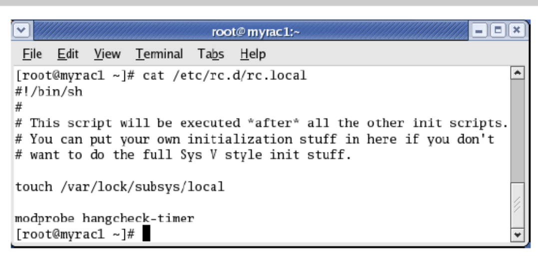


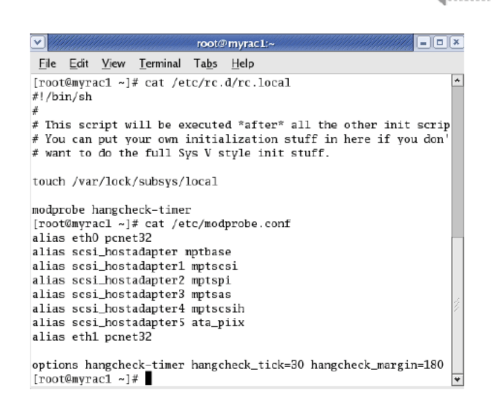
图 1-68 修改 rc.local 文件

02 配置 hangcheck-timer 参数，修改文件/etc/modprobe.conf，修改后的内容如图 1-69 所示。

03 重启系统，使得系统自动加载 hangcheck-timer 模块，指令如下所示。

```
[root@myrac2 ~] sync,sync,reboot
```

04 确认参数修改成功，系统自动加载了 hangcheck-timer 模块，如下例所示。



```
[root@myrac1 ~]# cat /etc/rc.d/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scrip
# You can put your own initialization stuff in here if you don'
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local

modprobe hangcheck-timer
[root@myrac1 ~]# cat /etc/modprobe.conf
alias eth0 pcnet32
alias scsi_hostadapter mptbase
alias scsi_hostadapter1 mptscsi
alias scsi_hostadapter2 mptspi
alias scsi_hostadapter3 mptsas
alias scsi_hostadapter4 mptscsih
alias scsi_hostadapter5 ata_piix
alias eth1 pcnet32

options hangcheck-timer hangcheck_tick=30 hangcheck_margin=180
[root@myrac1 ~]#
```

图 1-69　修改 modprobe.conf 文件

例子 1-5 查看系统是否加载了 hangcheck-timer 模块。

```
[root@myrac2~]$  grep hangcheck /var/log/messages | tail -1

Mar 16 20:36:59 myrac2 kernel: Hangcheck: starting hangcheck timer 0.5.0(tick
is 10 seconds,margin is 30 seconds).
```

输出结果说明已经启动了 hangcheck-timer 模块。

10. 按照系统需要对磁盘分区

分区的目的是配置裸设备，以及创建 ASM 磁盘，这里我们使用 fdisk 指令对/dev/sdb 物理磁盘进行分区。如下例所示，先介绍指令 fdisk 的参数。

例子 1-6 使用 fdisk 指令对/dev/sdb 物理磁盘进行分区。

```
[root@myrac1 ~] fdisk /dev/sdb

Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l    list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w  write table to disk and exit
   x   extra functionality ( experts only )
```

这里我们主要使用参数 n 和参数 w，参数 n 的含义是增加一个新分区，参数 w 的含义是创建磁盘分区，将分区信息写入分区表。下面创建分区/dev/sdb1 和/dev/sdb2，如下例所示。

例子 1-7 创建分区/dev/sdb1 和/dev/sdb2。

```
 [root@myrac1 ~] fdisk /dev/sdb

Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4):3
First cylinder (114-522,default 114):
Using default value 114
Last cylinder or +size or + sizeM or +sizeK (114-522,default 522):+1600m

Command (m for help): n
Command action
e   extended
p primary partition (1-4)
p
Selected partition 4
First cylinder (310-522 ,default 310):
Using default value 310
Last cylinder or +size or +sizeM or +sizeK (310-522, default 522): +1600m

Command (m for help): w
The partition table has been altered!

Calling ioctl( ) to re-read partition table.
Syncing disks.
[root@myrac1 ~]#
```

接着创建分区/dev/sdb3 和/dev/sdb4，目的是创建 ASM 逻辑卷需要的物理磁盘分区：VOL1 和 VOL2。

在节点 myrac1 上执行如下操作，创建物理磁盘分区/dev/sdb3 和/dev/sdb4，如下例所示。

例子 1-8 创建物理磁盘分区/dev/sdb3 和/dev/sdb4。

```
[root@myrac1 ~] fdisk /dev/sdb

Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4):3
First cylinder (114-522,default 114):
Using default value 114
Last cylinder or +size or + sizeM or +sizeK (114-522,default 522):+4000m

Command (m for help): n
Command action
e   extended
p primary partition (1-4)
```

```
p
Selected partition 4
First cylinder (310-522 ,default 310):
Using default value 310
Last cylinder or +size or +sizeM or +sizeK (310-522, default 522): +4000m

Command (m for help): w
The partition table has been altered!

Calling ioctl( ) to re-read partition table.
Syncing disks.
[root@myrac1 ~]#
```

通过 fdisk –l 指令查看分区结果，显示结果如图 1-70 所示。

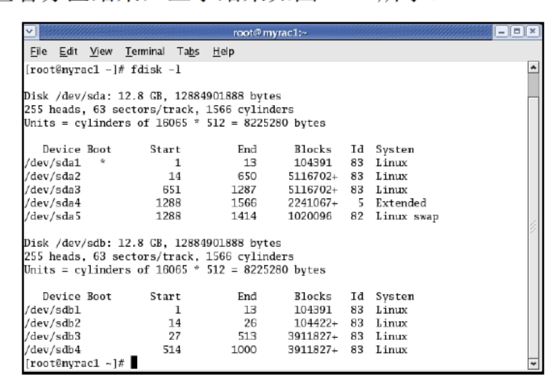


图 1-70　查看/dev/sdb 磁盘分区结果

11. 配置裸设备

裸设备用于存储 Clusterware 的 OCR 文件和 Voting Disk 文件。其中 OCR 文件存储各个节点的配置信息，修改集群中的任何一个节点，其实都是修改同一个文件。Voting Disk 文件存储了节点中成员的状态信息。

裸设备是指通过字符方式访问的设备，该种设备在读写数据时不需要缓冲区，在 Linux 环境下默认不提供裸设备服务，必须手动配置裸设备服务，同时需要修改裸设备访问的权限。配置裸设备需要以下几个步骤，并且在两个节点上都要执行。

01 编辑 /etc/sysconfig/rawdevices 文件，在文件中增加如下两行配置。

```
/dev/raw/raw1   /dev/sdb1
/dev/raw/raw2   /dev/sdb2
```

其中/dev/raw/raw1 为裸设备名，而对应的/dev/sdb1 为块设备名，也就是我们对磁盘/dev/sdb 的两个分区名。

02 修改裸设备的访问权限，编辑/etc/udev/permission.d/50-udev.permissions。修改的部分如下：

```
raw*:oracle:dba:0660
raw/*:oracle:dba:0660
```

03 启动裸设备服务，如下例所示。

例子 1-9 启动裸设备服务。

```
[root@myrac1 ~]# service rawdevices restart
Assigning devices:
          /dev/raw/raw1  -- >  /dev/sdb1
/dev/raw/raw1: bound to major 8,minor 17
          /dev/raw/raw2  -- >  /dev/sdb2
/dev/raw/raw2: bound to major 8, minor 18
done
```

以上输出表示，裸设备启动成功，其中/dev/raw/raw1: bound to major 8,minor 17 表示裸设备/dev/raw/raw1 绑定地址，即 Linux 系统在定位该设备时需要一个地址表示。

12. 创建 ASM 逻辑磁盘

这里我们使用 ASMLib 方式来创建 ASM 磁盘，此时需要首先安装 ASMLib RPM 包。我们首先需要确认当前 Linux 系统的版本来确认需要安装的 ASMLib RPM 包的版本，所以需要通过如下指令查找当前系统的版本。

例子 1-10 查询当前系统的版本。

```
[root@myrac1 ~]# uname -a
Linux myrac1 2.6.9-22.Elsmp #1 SMP Mon Sep 19 18:32:14 EDT 2005 i686 i686 i386 G
NU/Linux
```

这里所需要的 ASMLib RPM 包的版本必须与 Linux 的版本“2.6.9-22.Elsmp”一致。

以下是需要安装的 ASMLib RPM 包。

```
Oracleasm-support-2.0.3-1.i386.rpm
Oracleasmlib-2.0.2-1.i386.rpm
Oracleasm-2.6.9-22.Elsmp-2.0.3-1.i686rpm
```

下面演示如何安装这三个软件包，如图 1-71 所示。

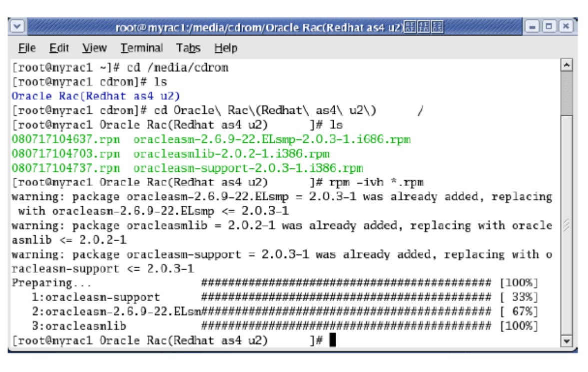


图 1-71 安装 ASMLib RPM 包



笔者此时是将所需要的 RPM 包制作成一个.ISO 文件，然后通过 Linux 的光驱加载该文件。

然后，我们进入创建 ASM 磁盘的实质操作步骤，在集群的一个节点创建 ASM 磁盘即可，在启动节点可以通过磁盘扫描的方式发现该 ASM 磁盘，具体步骤如下所示：

01 配置 ASM，执行 oracleasm configure 指令。

```
    [root@myrac1 ~] /etc/init.d/oracleasm configure
```

02 在节点 myrac1 启动创建 ASM 磁盘，如下例所示。

例子 1-11 创建 ASM 磁盘。

```
[root@myrac1 ~] /etc/init.d/oracleasm createdisk VOL1 /dev/sdb3
Marking disk "/dev/sdb3" as an ASM disk: [root@myrac1 ~]#
[root@myrac1 ~] /etc/init.d/oracleasm createdisk VOL2 /dev/sdb4
Marking disk "/dev/sdb4" as an ASM disk: [root@myrac1 ~]#
```

03 在节点 myrac2 扫描并查看在节点 myrac1 创建的 ASM 磁盘，如下例所示。

例子 1-12 扫描节点 myrac1 上创建的 ASM 磁盘。

```
[root@myrac2 ~] /etc/init.d/oracleasm  scandisks
Scanning system for ASM disks:                          [  OK  ]
[root@myrac2 ~] /etc/init.d/oracleasm  listdisks
VOL1
VOL2
[root@myrac2 ~]
```

从上述输出可以看出，节点 myrac2 可以通过 oracleasm 指令扫描并发现在节点 myrac1 创建的 ASM 磁盘 VOL1 和 VOL2。

1.6.7　安装 Clusterware

在配置完主机后，安装 Clusterware 的主机环境已经设置好，接下来安装 Clusterware 软件，我们运行该软件目录下的 runInstaller 程序。笔者将该 Clusterware 集群件软件拷贝到/oracle 目录下，并运行安装程序，如图 1-72 所示。

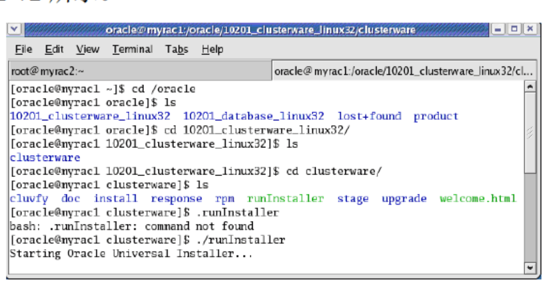


图 1-72　安装 Clusterware 集群体

运行界面如图 1-73 所示。

单击图 1-73 所示的“Next”按钮，进入如图 1-74 所示的对话框。

在图 1-74 中，会自动在 ORACLE_BASE 目录后创建 oraInventory 子目录，自动获得组名 dba，我们都接受默认设置。单击“Next”按钮，如图 1-75 所示。

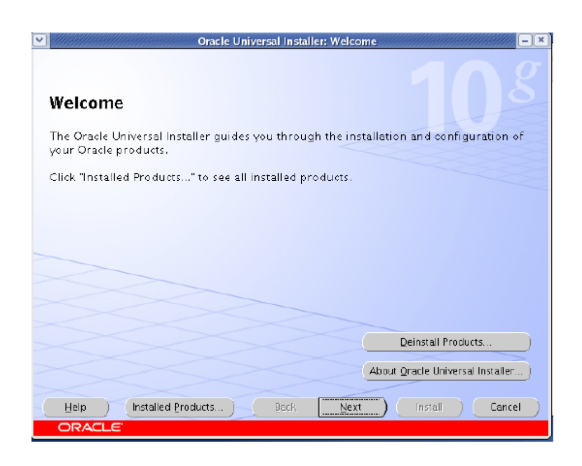


图 1-73　安装 Clusterware

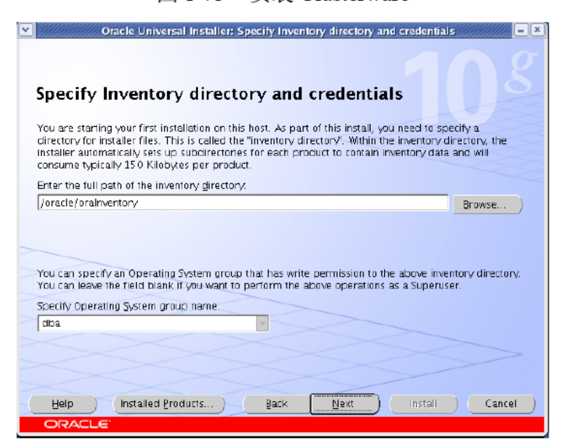


图 1-74　设置 Inventory 目录



图 1-75　设置 CRS 的安装路径

在图 1-75 中，要求输入安装 CRS 的路径名称以及安装 CRS 的全路径，此时，我们输入在编辑.bash_profile 文件时设置的 CRS_HOME 环境变量名以及对应的变量值。单击“Next”按钮，如

图 1-76 所示。

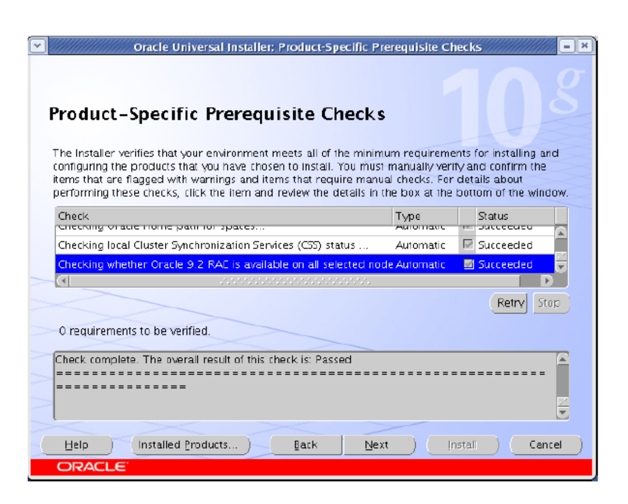


图 1-76　先决条件检查

图 1-76 的过程进行先决条件检查，必须保证不能出现 error，如果出现则需要通过提示说明修改错误或者修改配置之后，再 Retry，直到没有错误为止。单击“Next”按钮，如图 1-77 所示。

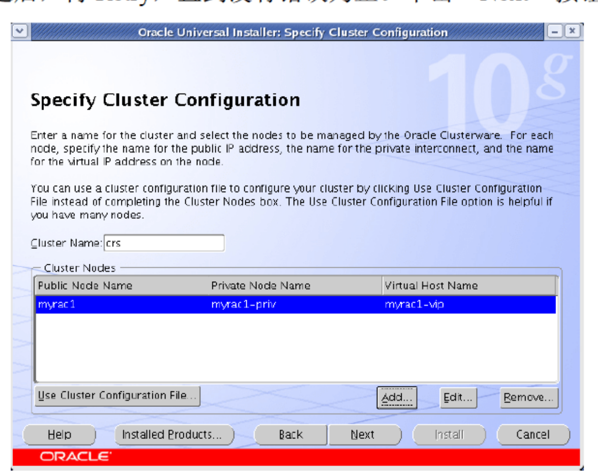


图 1-77　自动发现一个节点

图 1-77 中的 Cluster Nodes 信息是自动发现的，这里需要添加集群的另一个节点 myrac2，单击图 1-77 的“Add”按钮，如图 1-78 所示。

在图 1-78 中输入第二个节点 myrac2 的 Public 节点名、Private 节点名以及 Virtual 主机名。然后单击“OK”按钮，回到图 1-79 所示的窗体中。

图 1-79 中的 Cluster Nodes 已经包括集群中的所有节点了，这样 Clusterware 就知道集群中有几个节点的相关信息。单击图 1-79 的“Next”按钮，如图 1-80 所示。

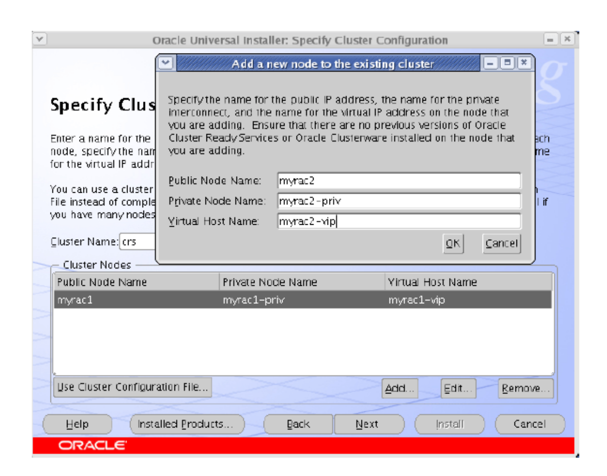


图 1-78　加入新的节点

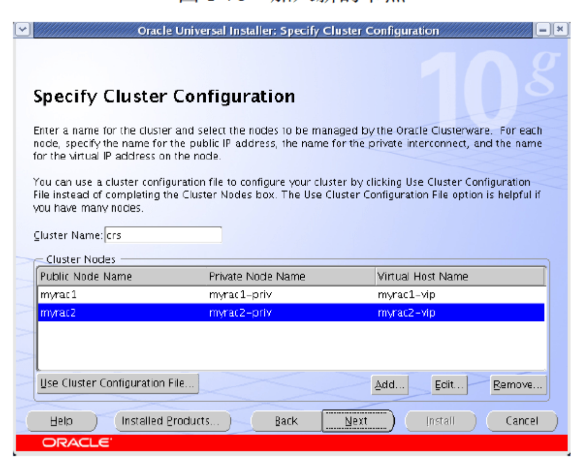


图 1-79　定义两个节点

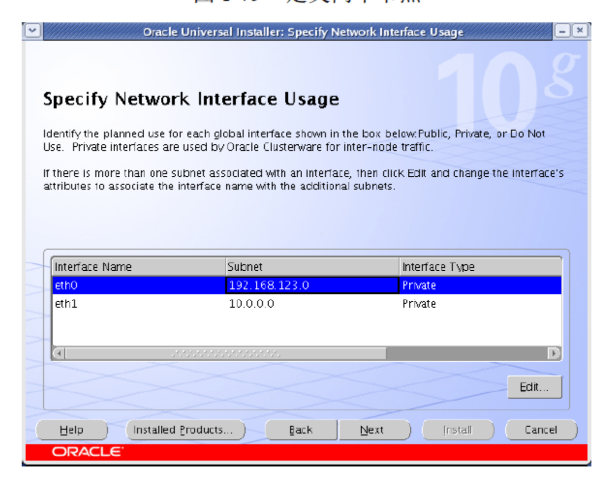


图 1-80　设置 Public 接口 IP

在图 1-80 中，单击“Edit”按钮，编辑网络接口，如图 1-81 所示。

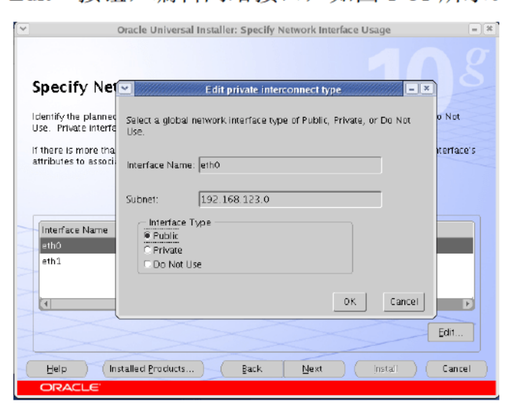


图 1-81　设置 Public 网络接口

在图 1-81 中，选择 eth0 作为 Public 接口，单击“OK”按钮回到图 1-82 所示的窗体。

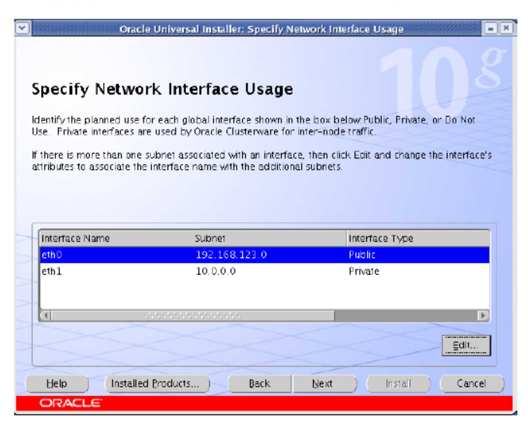


图 1-82　设置网络接口

单击图 1-82 所示的“Next”按钮，进入如图 1-83 所示的窗体。

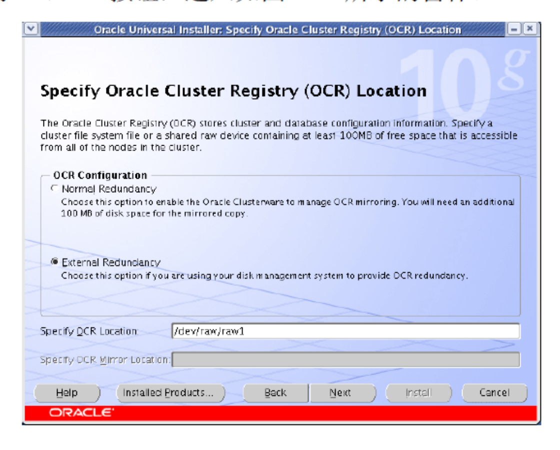


图 1-83　设置 OCR 存储

在图 1-84 所示的界面中，设置 OCR 的存储区，我们选择“External Redundancy”选项，通过裸设备作为 OCR 的存储区。

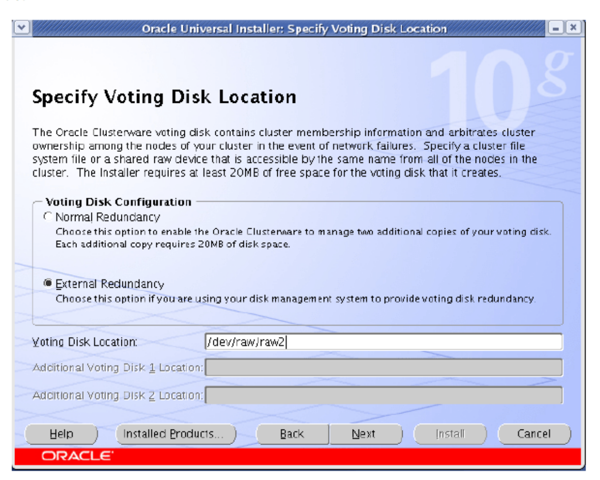


图 1-84　设置 Voting Disk 存储

在图 1-84 中，设置 Voting Disk 的存储区，我们选择“External Redundancy”选项，通过裸设备作为 OCR 的存储区。再单击“Next”按钮，到如图 1-85 所示的界面，单击“Install”按钮，打开如图 1-86 所示的界面。

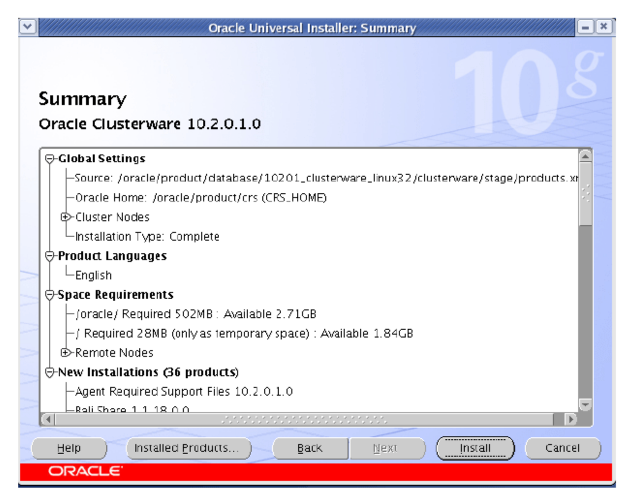


图 1-85　集群件安装信息汇总

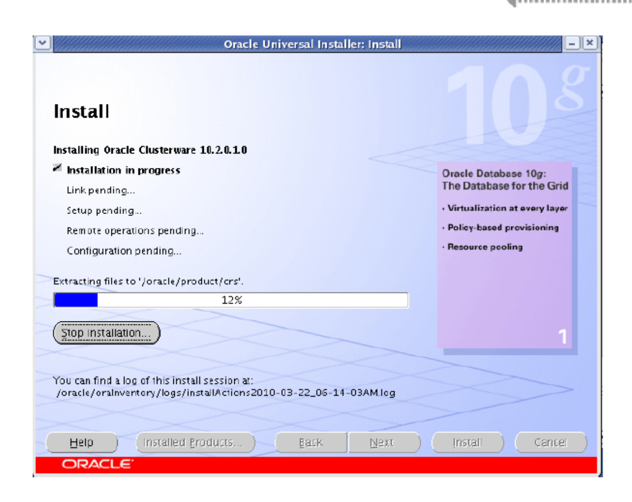


图 1-86　安装 Clusterware 软件

在图 1-86 的 Clusterware 软件安装完毕后，会弹出如图 1-87 所示对话框。需要使用 root 用户分别在两个节点执行两个脚本文件，一个为/oracle/oraInventory/orainstRoot.sh，一个为/oracle/product/ crs/root.sh。

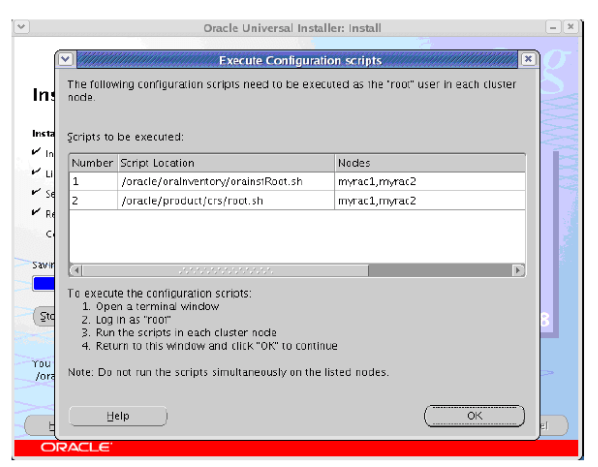


图 1-87　提示执行脚本文件

我们在节点 myrac1 执行脚本文件 orainstRoot.sh，如图 1-88 所示。

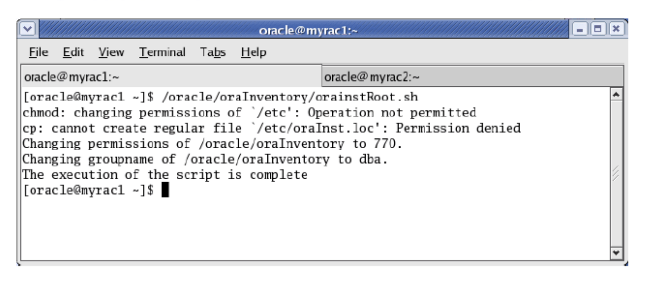


图 1-88　执行 oraInstRoot.sh 文件

在节点 myrac1 继续执行脚本文件 root.sh，如图 1-89 所示。

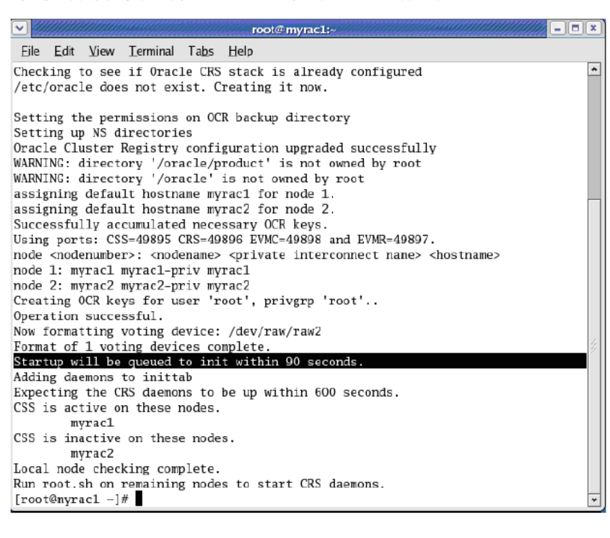


图 1-89　在节点 myrac1 执行脚本文件 root.sh

在节点 myrac2 执行 orainstRoot.sh 文件的内容与在 myrac1 的执行结果相同，这里给出子节点 myrac2 执行 root.sh 脚本文件的结果，如图 1-90 所示。

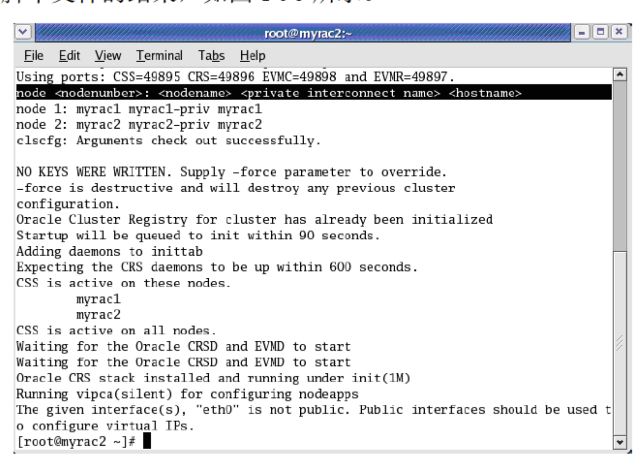


图 1-90　节点 myrac2 执行 root.sh 脚本

在两个节点 myrac1 和 myrac2 执行完两个脚本后，返回图 1-79 所示的界面，单击“OK”按钮，进入如图 1-91 所示的界面。

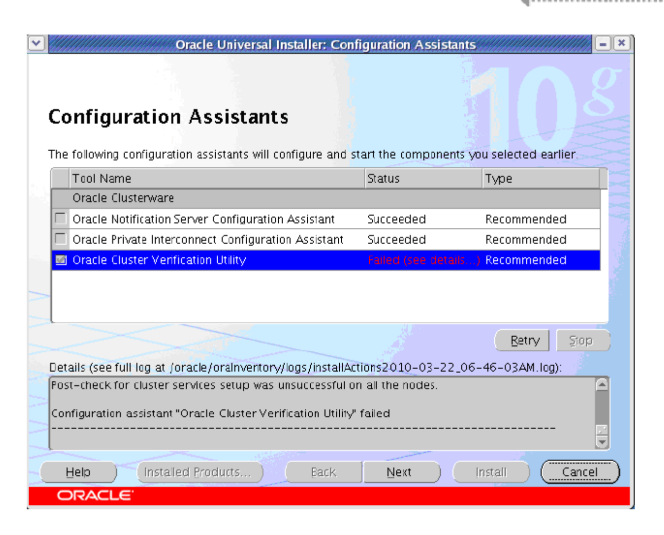


图 1-91　配置检查

图 1-91 中会进行配置检查工作，如果出现问题，如 css 资源没有启动等，会弹出如图 1-92 所示的对话框。

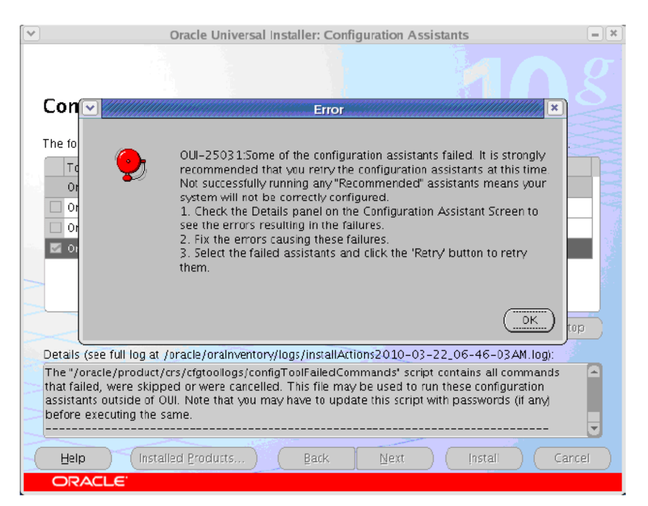


图 1-92　检测到配置错误

图 1-92 提示 OUI-25031 错误。解决错误的办法是回到节点 myrac2，使用 VIPCA 启动 gsd ons vip 资源，并将这些资源注册到 CRS 中，执行指令如下所示。

```
[root@myrac2 ~] # /oracle/product/crs/bin/vipca
```

运行该指令弹出如图 1-93 所示的 VIP 配置窗口。

单击“Next”按钮到下一个配置窗口，如图 1-94 所示。在图 1-94 中所示的窗体选择支持 VIP 的网络接口，显然只有 etho 为 Public 接口，默认是选中该接口，再依次单击“Next”按钮，出现的窗体如图 1-95、图 1-96 所示。

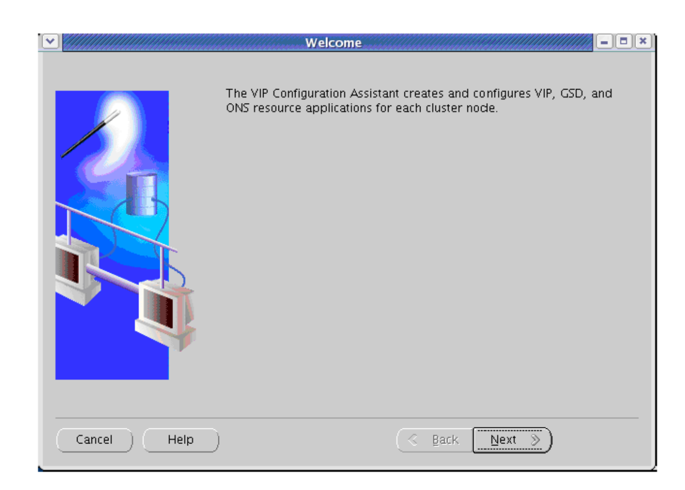


图 1-93 VIP 配置窗口

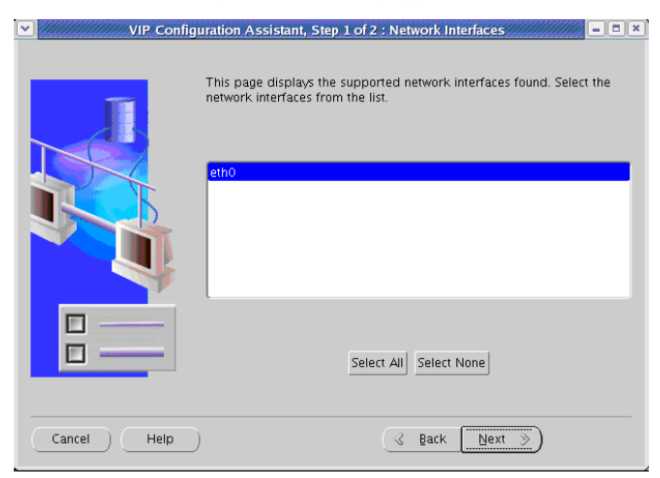


图 1-94 选择 VIP 绑定端口

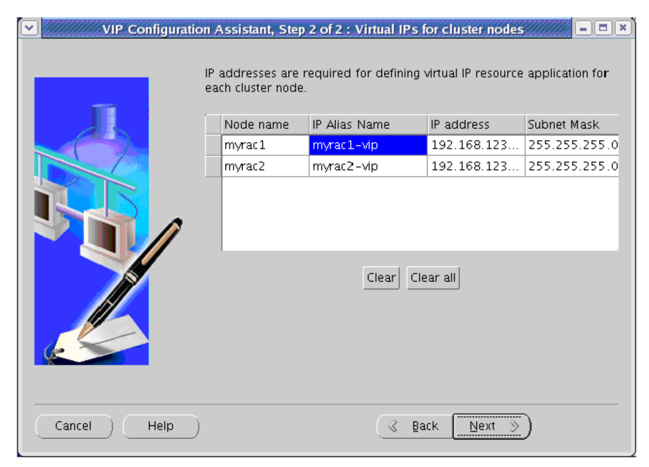


图 1-95 设置 VIP 地址和别名

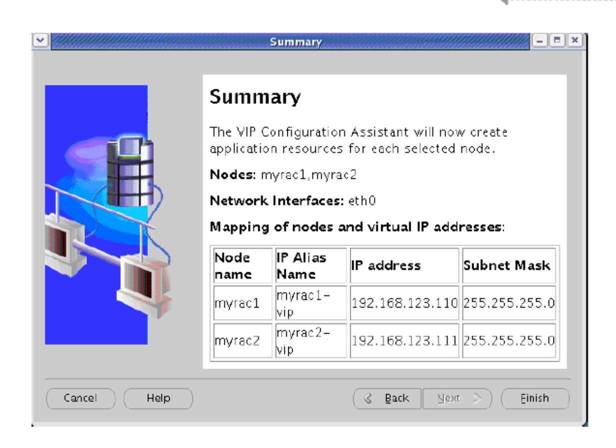


图 1-96　VIP 配置汇总信息

单击图 1-96 所示的“Finish”按钮，则弹出如图 1-97 所示的对话框，提示创建资源以及启动资源应用。

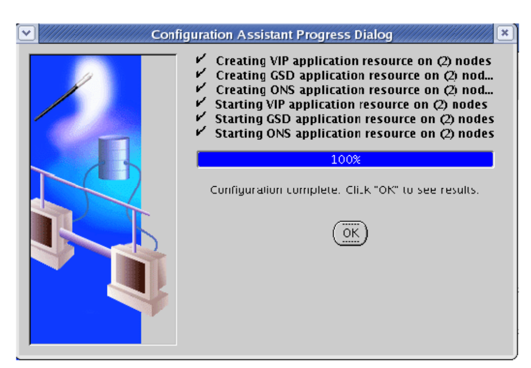


图 1-97　创建资源及启动应用

一旦图 1-97 所示的创建和启动任务顺利完成，则弹出如图 1-98 所示的对话框，给出配置结果的汇总信息。

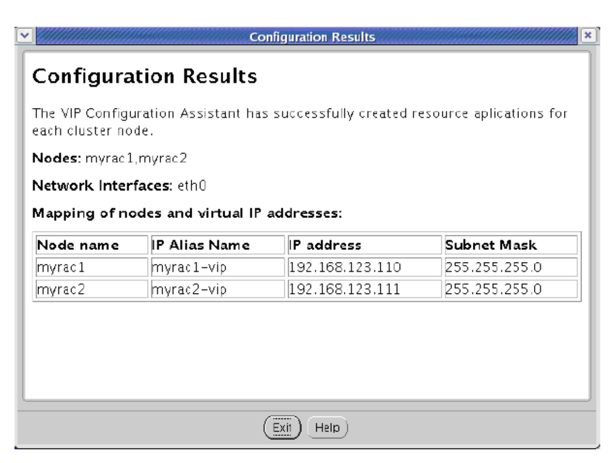


图 1-98　使用 VIPCA 配置汇总

配置完 VIPCA 后，再重新执行安装后的验证工作，则会顺利通过检测，完成 Clusterware 的安装。

在安装完毕后，我们通过如下指令判断当前 RAC 环境下的两个节点的 CRS 资源是否启动运行，如图 1-99 所示。

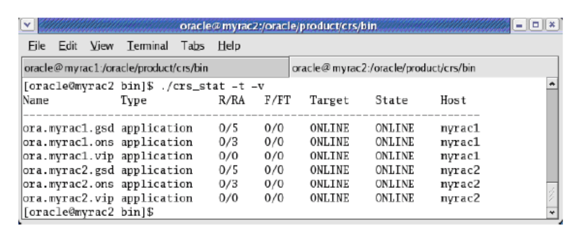


图 1-99　查看注册到 CRS 的各种资源

1.6.8　安装数据库软件

将下载的数据库安装软件解压缩到 Linux 系统的某个目录下，执行文件中的 runInstaller 可执行程序。启动界面如图 1-100、图 1-101 所示。

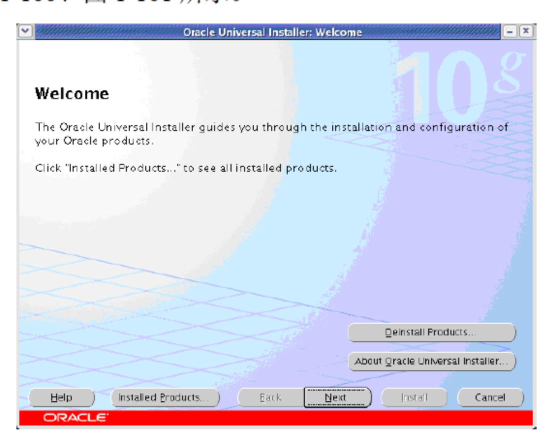


图 1-100　欢迎界面

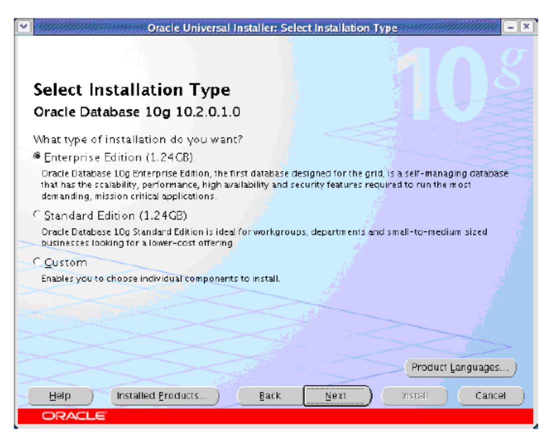


图 1-101　选择安装数据库类型

在图 1-101 所示的界面中选择安装数据库的类型，接着单击“Next”按钮进入下一个窗口，如图 1-102 所示，设置数据库管理软件的全目录。

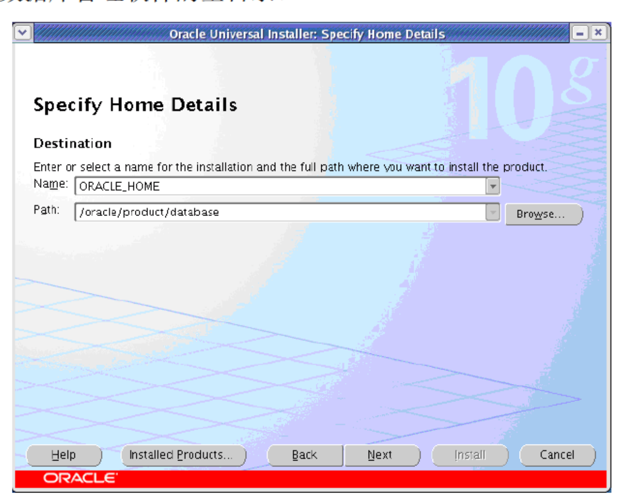


图 1-102　设置 ORACLE_HOME 路径

单击图 1-102 的“Next”按钮进入下一个窗口，如图 1-103 所示，DBCA 自动探测到集群节点的存在，选中所有节点。

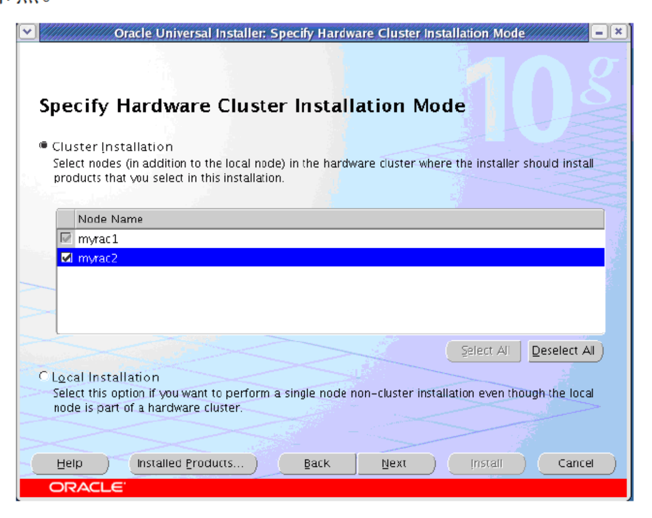


图 1-103　自动探测到 RAC 环境，选择安装节点

在图 1-104 所示的界面中预检测完毕后，进入选择配置选项界面，会提示需要安装的软件，可以选择创建数据库、创建 ASM 存储或者安装数据库管理软件，如图 1-105~图 1-107 所示。

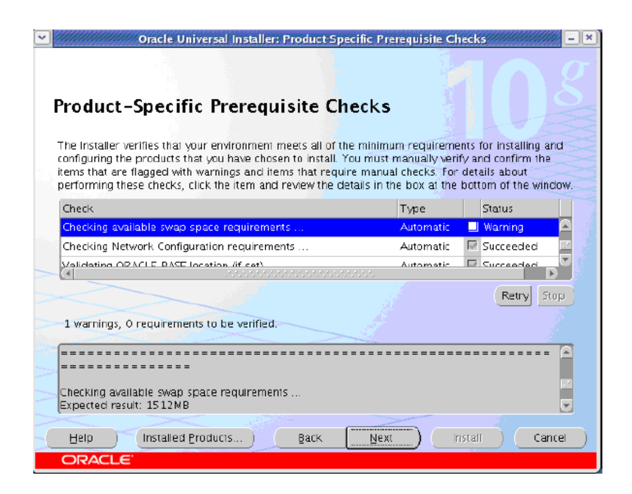


图 1-104　预检验过程

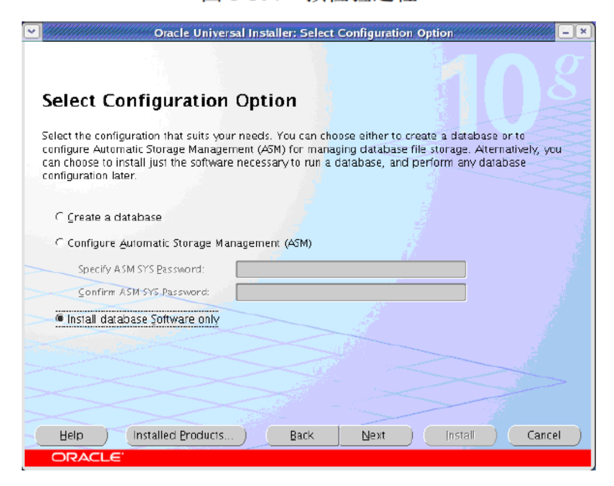


图 1-105　选择只安装数据库管理软件及 RDBMS



图 1-106　数据库安装信息汇总

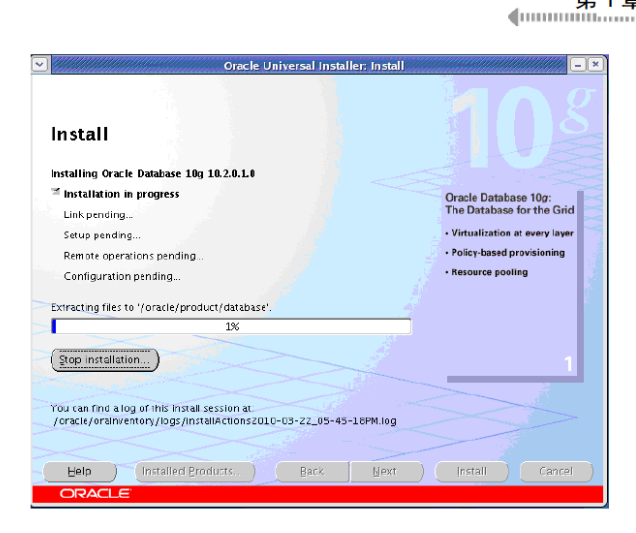


图 1-107　安装数据库管理软件

在数据库软件安装完毕后，会弹出图 1-108 所示的对话框，要求在每个节点执行脚本。

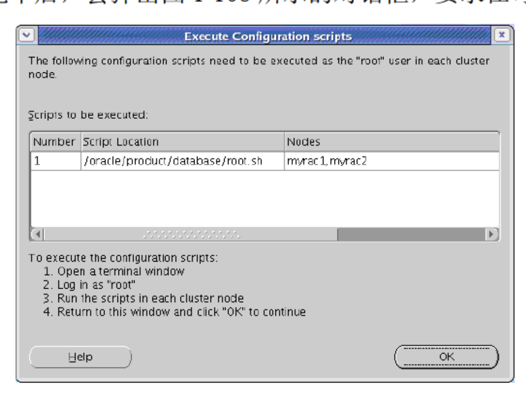


图 1-108　需要执行的安装脚本

下面，我们在节点 myrac1 执行该脚本，同样在节点 myrac2 执行同样的脚本，结果相同。注意，必须使用 root 用户执行该脚本，如图 1-109、图 1-110 所示。

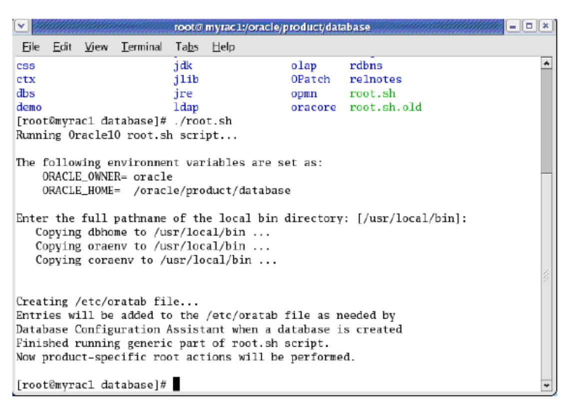


图 1-109　脚本的执行过程

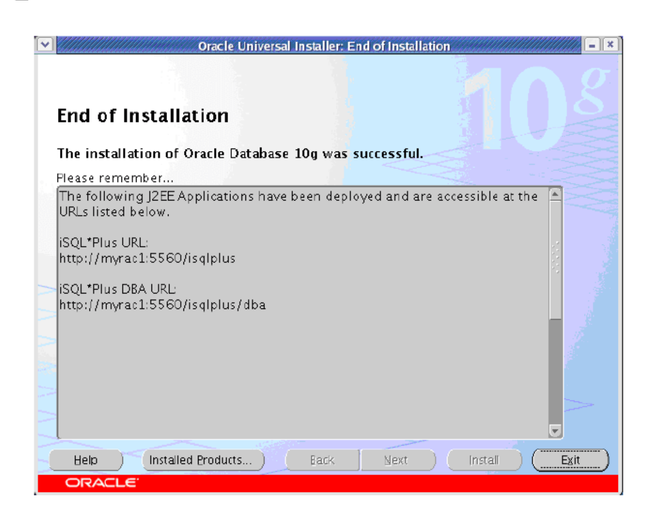


图 1-110　数据库安装结束

在安装完数据库管理软件后，我们就有了 DBCA 工具来创建 ASM，以及创建基于 ASM 存储的数据库。

1.6.9　启动监听

如果是在单机上安装并使用 RAC 集群环境下的数据库，可以不需要配置监听，因为他通过内部的 IPC 通信实现连接。如果在生产数据库中，当然需要启动监听来等待客户的连接请求，这里我们演示如何启动监听器。

使用 DBCA 来启动配置助手，如图 1-111 所示，选择集群配置就可以从两个集群节点上同时启动监听。单击“Next”按钮，如图 1-112 所示，这里选中两个节点。单击“Next”按钮，打开窗体如图 1-113 所示。

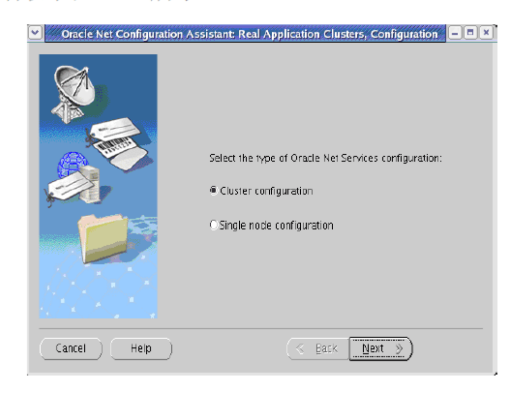


图 1-111　选择网络服务类型，选择集群配置

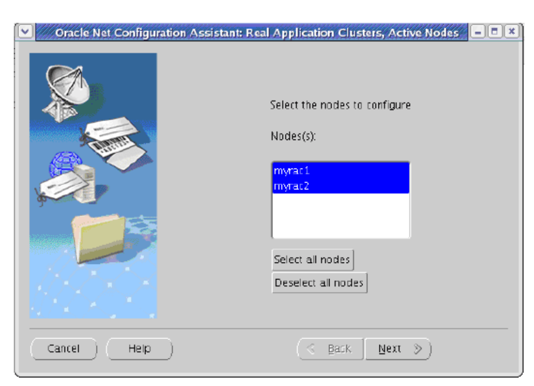


图 1-112　选择集群中的节点

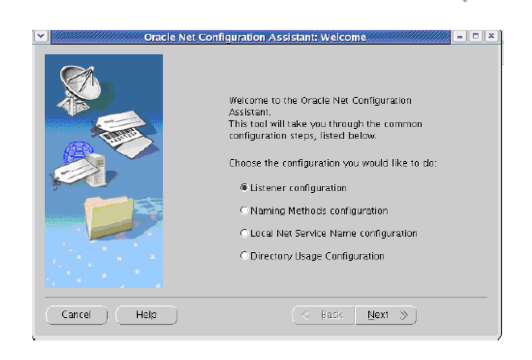


图 1-113　选择配置 Listener

在图 1-113 所示的窗体中选择监听器配置。单击“Next”按钮，如图 1-114 所示，选中“Add”单选按钮，单击“Next”按钮，打开新的配置窗口如图 1-115 所示。

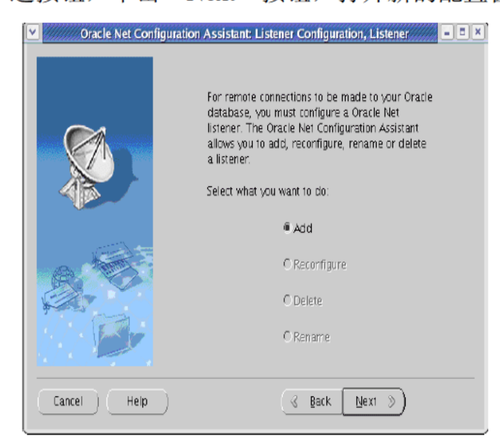


图 1-114　新增监听

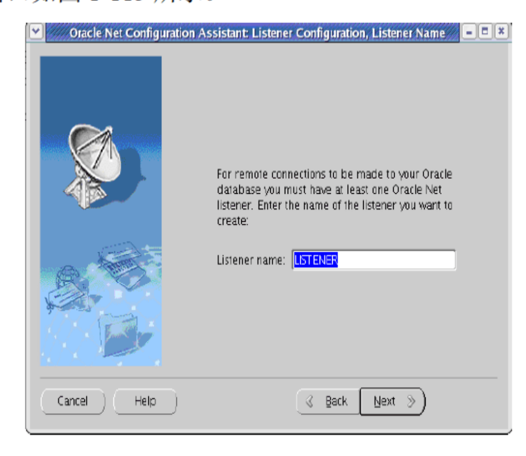


图 1-115　设置监听器名称

在图 1-115 所示的窗体中输入监听器名字，单击“Next”按钮打开新的配置窗口，如图 1-116 所示，选择监听器使用的通信协议为 TCP（一种面向连接的传输控制协议）。

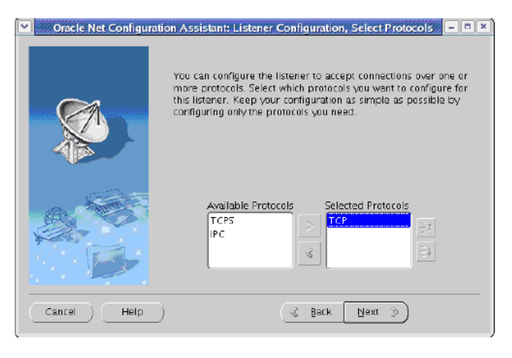


图 1-116　选择监听器使用的协议

单击图 1-116 中的“Next”按钮打开新的配置窗口，如图 1-117 所示，选择监听端口，即数据库服务在哪个端口等待用户连接请求，默认是 1521，这里采用默认设置。

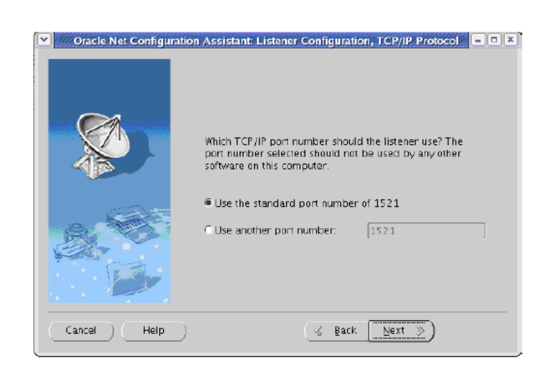


图 1-117　选择监听端口

继续单击图 1-117 的“Next”按钮打开新的配置窗口，如图 1-118 所示，这里不必再配置监听器了。选择“NO”选项，再单击“Next”按钮，结束配置，如图 1-119 所示。

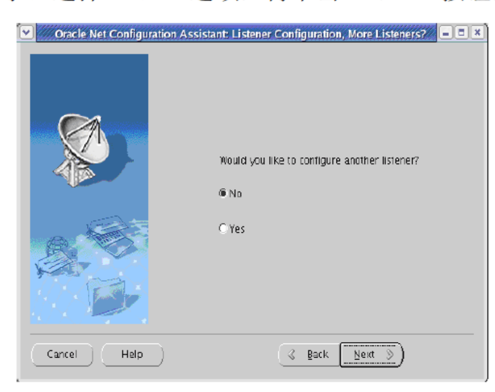


图 1-118　结束监听器配置

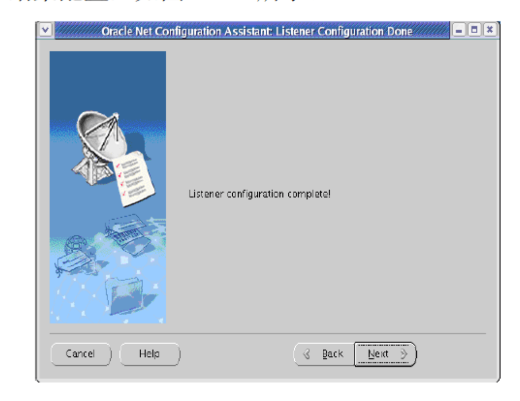
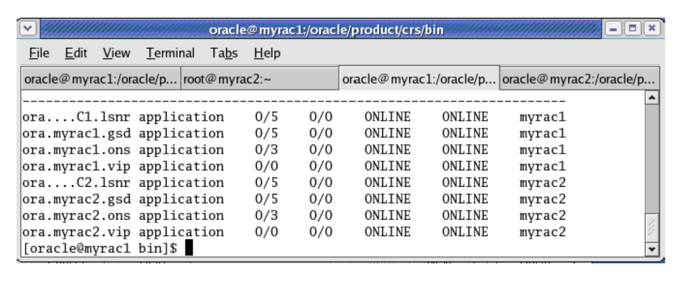


图 1-119　结束配置

配置完监听器之后，可以查看监听器资源的当前状态，使用 crs_stat –t –v 指令查看当前的注册到 CRS 中的资源状态，如图 1-120 所示。




```
----------------------------------------------------------------------
ora....C1.lsnr application    0/5    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.gsd application    0/5    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.ons application    0/3    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.vip application    0/0    0/0    ONLINE    ONLINE    myrac1
ora....C2.lsnr application    0/5    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.gsd application    0/5    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.ons application    0/3    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.vip application    0/0    0/0    ONLINE    ONLINE    myrac2
[oracle@myrac1 bin]$
```

图 1-120　验证监听服务

在配置启动监听器之后，CRS 会记录资源信息和状态，并监控这些数据库相关资源的状态，我们通过图 1-120 所示的结果说明监听资源运行正常，即 ora…C1.lsnr 和 ora…C2.lsnr 状态都为 ONLINE。

1.6.10　创建 ASM

下面我们创建 ASM，创建 ASM 磁盘组并启动 ASM 实例。我们使用 DBCA 工具来配置 ASM，启动 DBCA 的指令如下所示。

```
[root@myrac1 ~] # /oracle/product/database/bin/dbca
```

DBCA 启动后，选中“Oracle Real Application Clusters databases”单选按钮，如图 1-121 所示。我们依次单击“Next”按钮，步骤依序如图 1-121、图 1-122 所示。

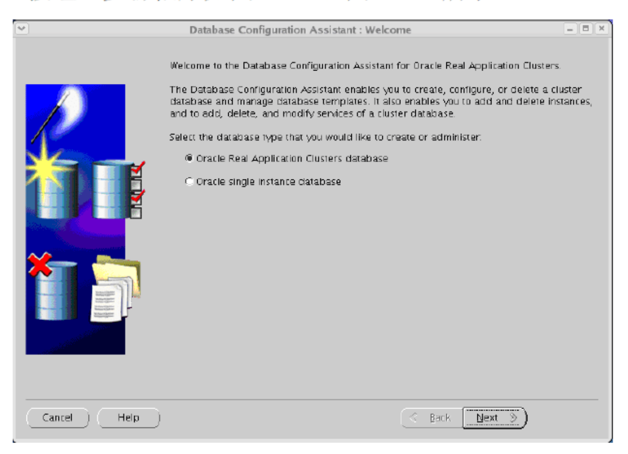


图 1-121　选择集群数据库

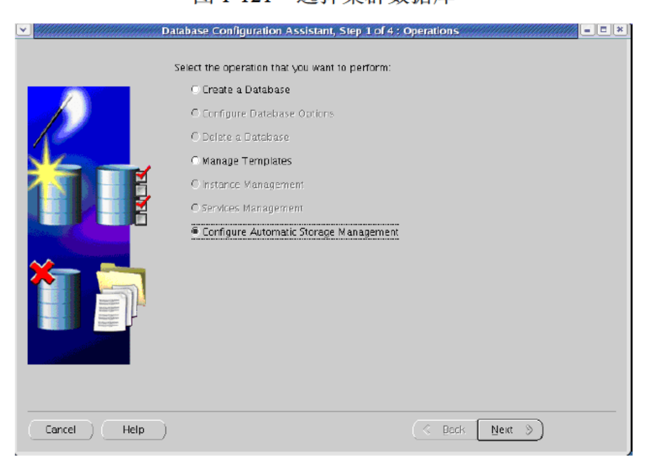


图 1-122　选择配置 ASM

为这些节点创建 ASM 磁盘组，即这两个节点共享 ASM 磁盘文件，操作顺序如图 1-123~图 1-126 所示。

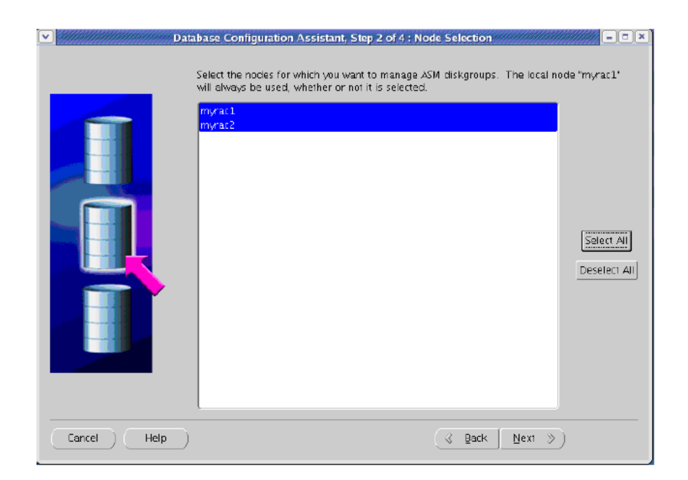


图 1-123　选择创建 ASM 磁盘组的节点

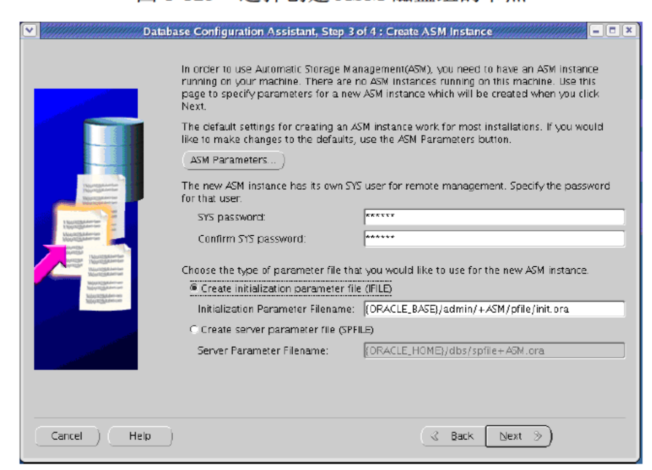


图 1-124　设置 ASM 实例密码以及参数文件位置

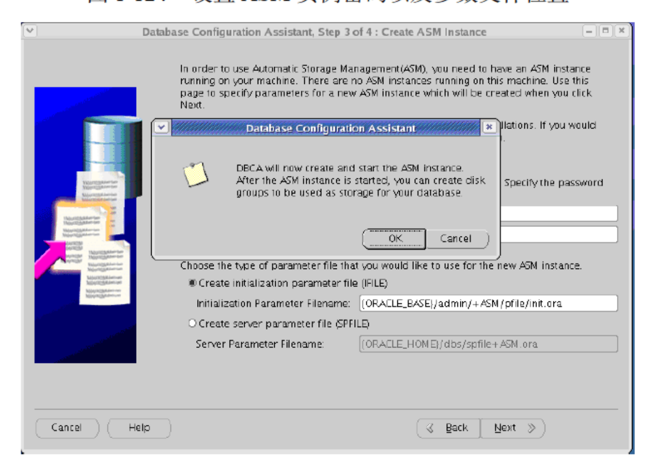


图 1-125　提示创建并启动实例

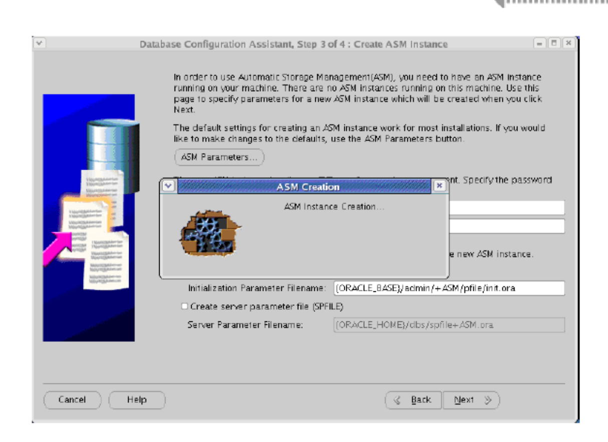


图 1-126　创建 ASM 实例

单击图 1-127 所示窗体中的“Create New”按钮，创建磁盘组，如图 1-128 所示，输入磁盘组名称，子冗余策略中选择 External，选择 ASM 磁盘来创建磁盘组。

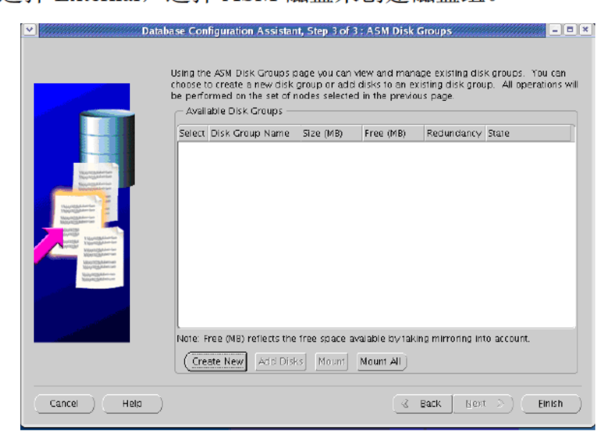


图 1-127　创建 ASM 磁盘组

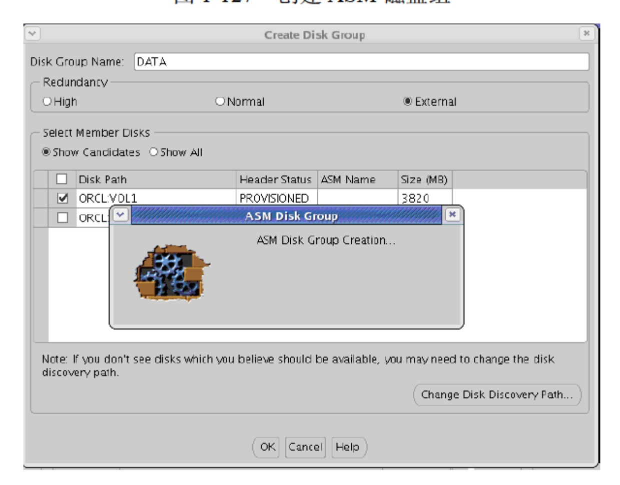


图 1-128　创建磁盘组

在成功创建磁盘组之后，实例已经创建并自动启动了，而实例的信息也同样作为一种资源注册到 Clusterware 的 CRS 中了。可以通过 crs_stat –t –v 指令查看“实例”资源的当前状态，如图 1-129 所示。

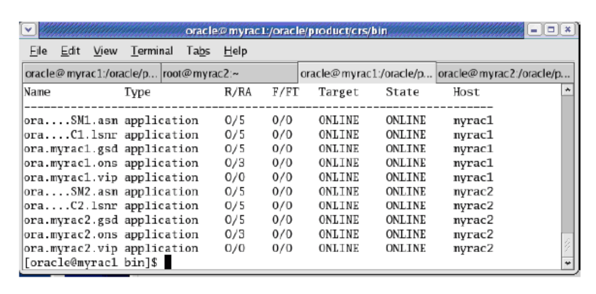


图 1-129　查看实例资源的运行状态

在图 1-129 中，我们看到资源 ora…SM1.asm 和 ora…SM2.asm 都是运行正常的，因为其 Target 和 State 都为 ONLINE。

如果需要我们还可以继续创建磁盘组，前提是必须有多余的 ASM 磁盘可用。当然如果确实需要，只要硬件许可，我们可以创建更多的 ASM 磁盘以满足系统对容量的需要，创建新的 ASM 磁盘组如图 1-130 所示。

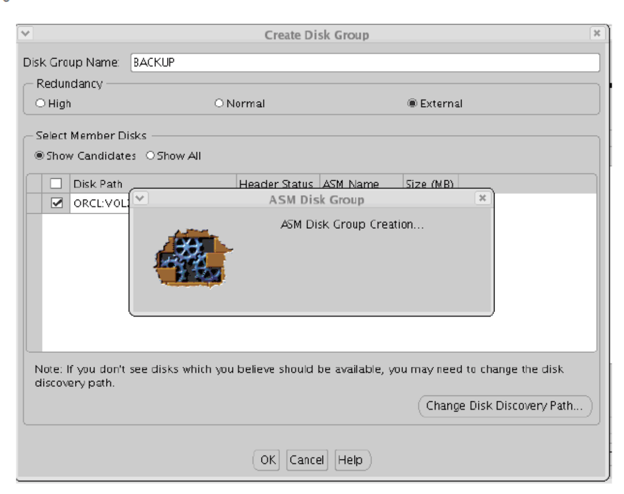


图 1-130　创建新的磁盘组

此时，我们已经安装好了 Clusterware 软件、数据库管理软件，启动了监听器以及创建了 ASM 实例和 ASM 磁盘组，下面就可以轻松地创建 Oracle 数据库了，我们将数据库文件全部安装在 ASM 磁盘组 DATA 上。

1.6.11　创建数据库

与创建 ASM 一样，我们启动 DBCA 来创建数据库，如图 1-131 所示，选中“Create a Database”单选按钮，然后单击“Next”按钮。下面我们依序用图说明，关键步骤再做描述，如图 1-132~图 1-134 所示。

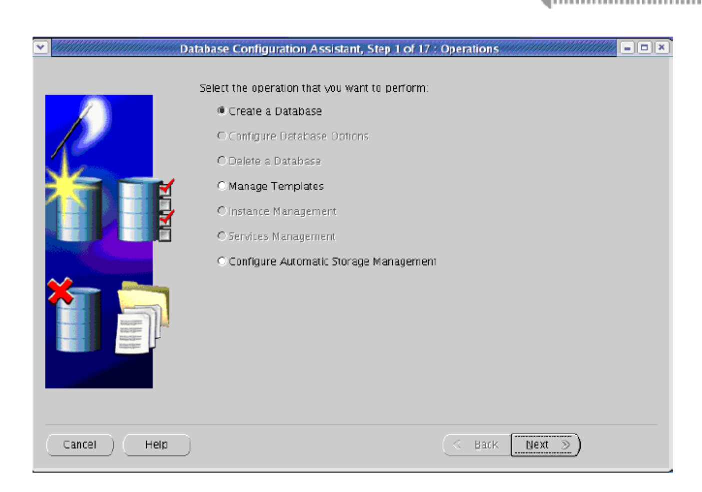


图 1-131　选择创建数据库

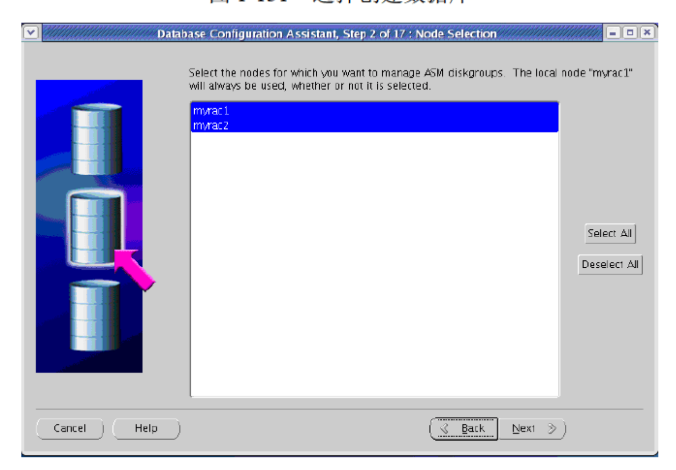


图 1-132　选择节点来管理 ASM 磁盘组

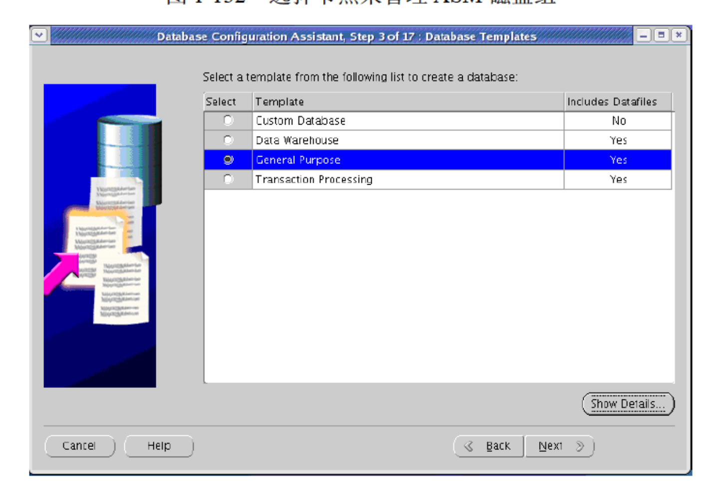


图 1-133　选择数据库类型

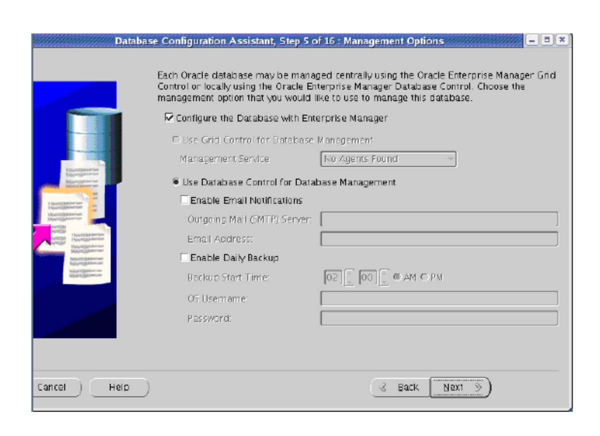


图 1-134 选择数据库管理方式为 EM

设置用户密码，可以如图 1-135 所示设置统一的密码，也可以使用不同的密码。如果希望为不同的用户设置不同的密码，则选中“Use Different Passwords”单选按钮。接下来的操作步骤如图 1-136~图 1-138 所示。

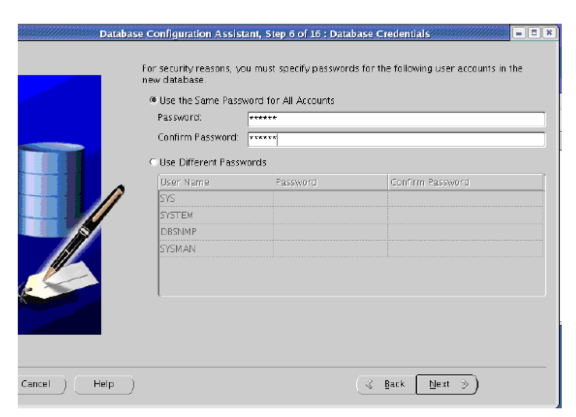


图 1-135 设置用户密码

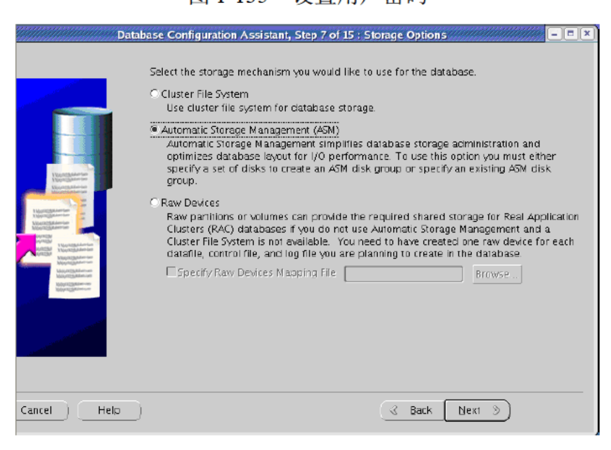


图 1-136 选择 ASM 存储管理数据库文件

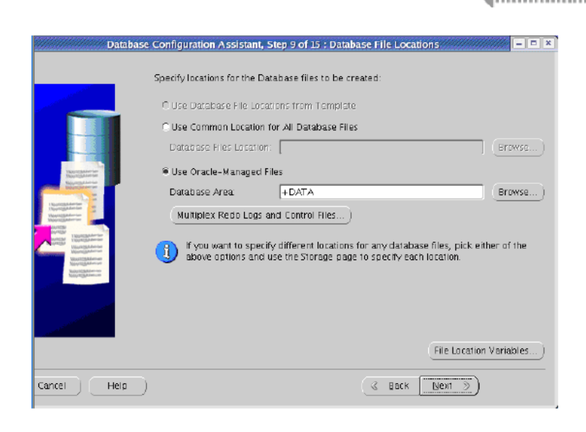


图 1-137　选择使用 ASM 磁盘组 DATA 来存储数据库文件

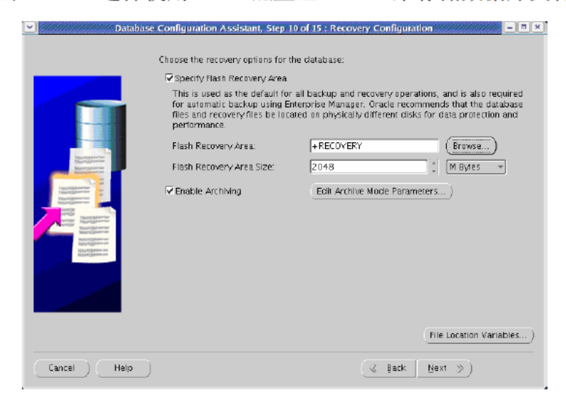


图 1-138　设置闪回恢复区以及启动归档模式

在该步骤中，闪回恢复区使用 ASM 磁盘组 RECOVERY，这样两个实例共用一个闪回恢复区，同时启动归档模式，归档目录采用默认路径，所以两个实例也共享一个归档文件。如果出于高可靠性的考虑，也可以在两个实例中分别设置归档目录，但是两个实例的归档文件才是一个完整的数据库归档文件。

下面接着设置内存分配，这里可以采用默认值，如图 1-139 所示。接下来的操作参看图 1-140、图 1-141 所示的窗体。

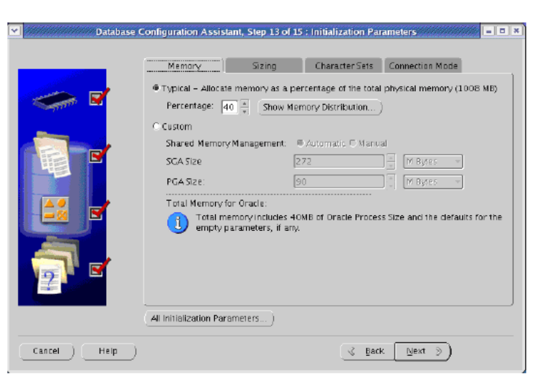


图 1-139　设置数据库内存等参数选项

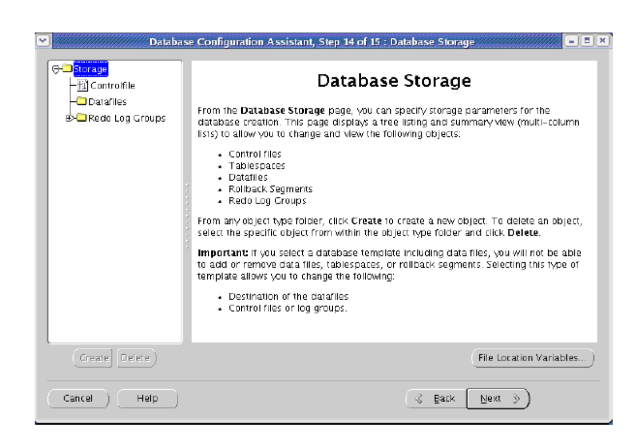


图 1-140　数据库存储设置信息汇总

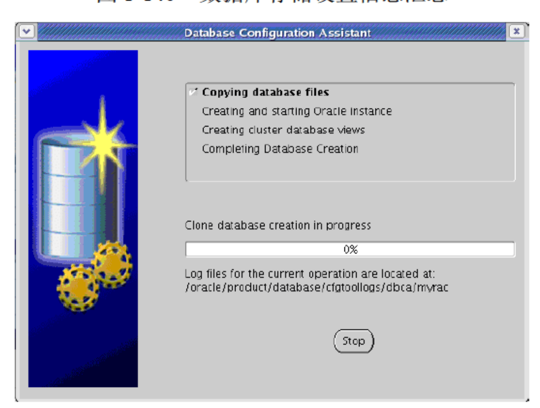


图 1-141　安装数据库并启动数据库实例

当数据库安装完毕后，与单实例数据库创建相同会自动启动数据库实例，所以安装的最后一个步骤就是启动数据库实例，此时数据库实例也会注册到 Clusterware 的 CRS 中，我们通过指令 crs_stat –t –v 来查看该资源的运行情况，如图 1-142 所示。

```
[root@myrac1 bin]# ./crs_stat -t -v
Name           Type        R/RA   F/FT   Target    State     Host
----------------------------------------------------------------------
ora.myrac.db   application 0/1    0/1    ONLINE    ONLINE    myrac2
ora....c1.inst application 0/5    0/0    ONLINE    ONLINE    myrac1
ora....c2.inst application 0/5    0/0    ONLINE    ONLINE    myrac2
ora....SM1.asm application 0/5    0/0    ONLINE    ONLINE    myrac1
ora....C1.lsnr application 0/5    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.gsd application 0/5    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.ons application 0/3    0/0    ONLINE    ONLINE    myrac1
ora.myrac1.vip application 0/0    0/0    ONLINE    ONLINE    myrac1
ora....SM2.asm application 0/5    0/0    ONLINE    ONLINE    myrac2
ora....C2.lsnr application 0/5    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.gsd application 0/5    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.ons application 0/3    0/0    ONLINE    ONLINE    myrac2
ora.myrac2.vip application 0/0    0/0    ONLINE    ONLINE    myrac2
```

图 1-142　数据库安装后的资源状态

显然从上例代码可以看出数据库实例资源运行正常，因为资源名 ora.myrac.db 运行正常，为 ONLINE 状态。

1.7 本章小结

RAC 就是由多个物理的计算机节点组成的数据库系统，逻辑上由 Clusterware 集群件、共享存储、网络组件以及 CRS 资源组成。部署 RAC 的数据库系统提高了系统的可靠性，实现了数据服务的业务负载。集群件 Clusterware 是 RAC 的核心软件，该软件管理 RAC 集群中的节点硬件资源，将多个节点虚拟成一个计算机，该软件由 OCR、Voting Disk、后台进程以及网络组件组成，了解了它的组成有助于理解其功能本质。

RAC 的部署也是本章的一个重点，本章我们需要掌握主机配置的信息，以及为什么这样配置，如 VIP 的配置等。主机的配置是个繁琐的过程，但是又是非常重要的内容，如果主机配置有问题则会造成安装过程错误频频。在主机配置完成后，我们继续介绍了部署 RAC 的环境所需的软件，以及具体软件的安装步骤。首先我们安装了 Clusterware 集群件软件，接着安装了数据库管理软件，启动节点的监听程序，创建数据库等步骤。其实，我们可以在创建数据库管理软件时创建数据库并同时启动监听程序，显然这样做虽然步骤简单，但是在 RAC 环境下，这样多的软件同时安装会大大提高出错的概率。最后，我们成功安装了一个 RAC 环境，这样在以后的学习中，我们就可以使用这个模拟环境了。

第 2 章

ASM自动存储管理

ASM 自动存储管理是 Oracle 推荐的一种智能化数据库文件存储管理工具，它采用 OMF 文件格式来创建目录文件，它通过自动管理磁盘组来平衡 I/O，通过镜像技术来实现数据文件的高可用性。在 RAC 环境下，Oracle 也推荐使用 ASM 作为共享存储，这样大大提高了 OracleDBA 对存储的把控能力。本章我们将介绍 ASM 的系统架构、实例架构、ASM 实例创建、ASM 实例管理，重点介绍 ASM 磁盘组管理和 ASM 文件。为了更好地理解本章的知识点，在 2.11 节给出一个例子，将一个基于本地文件系统的数据库迁移到 ASM 实例上去。

2.1　Oracle自动存储管理概述

对于一个大型的数据库，数据文件的维护或许会花去一个 DBA 大量的时间，因为要维护上千个数据文件显然不是一件轻松的事情。Oracle 的发展趋势也是向着“自动化”和“智能化”方向发展，自动存储管理 ASM 就是 ORACLE 对数据库文件进行自动管理的工具。

在创建数据库时会出现如图 2-1 所示的界面，其中就有一个选项，提示是创建基于本地管理的数据库还是基于 ASM 管理或者基于裸设备的数据库。其中，如果选择基于 ASM 的存储管理就会在 ASM 管理的逻辑磁盘组上，创建只有 ASM 认识的数据库文件（当然前提是已经创建了 ASM 磁盘组）。

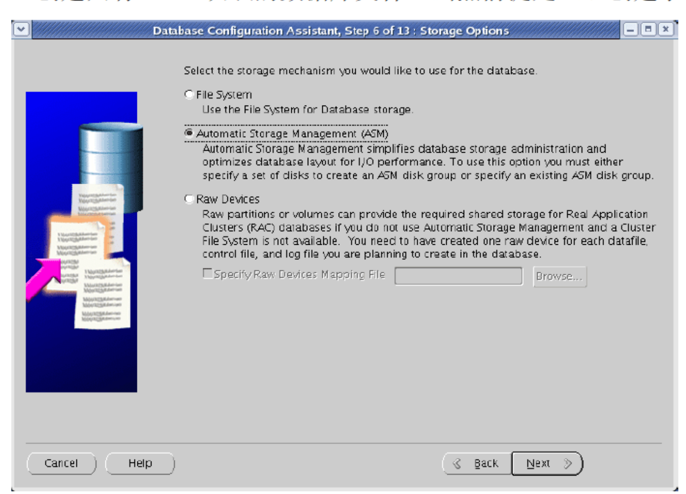


图 2-1　选择数据库存储机制

ASM 自动存储技术的功能类似于逻辑卷技术，但是它是 Oracle 的产品，所以不需要借助第三方工具实现磁盘文件的自动管理。为了减少 DBA 对于数据库文件的维护成本，ASM 通过镜像（一种通过冗余提供数据保护的技术）技术保护数据文件，通过条带化技术平衡磁盘 I/O，通过自动平衡机制动态管理磁盘，实施 ASM 摆脱了对第三方存储工具的使用，DBA 维护数据库就更容易。

ASM 自动存储和数据库管理系统 RDBMS 的关系如图 2-2 所示。

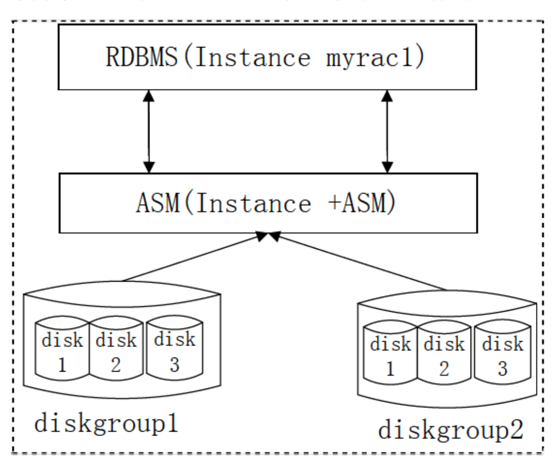


图 2-2　ASM 实例与 RDBMS 实例之间的关系

在图 2-2 中可以看出，在 RDBMS 和数据库物理文件之间需要 ASM 实例的参与，也就是在启动并使用数据库之前，必须先启动 ASM 实例，否则数据库由于找不到相关的文件而无法启动，需要说明的是一个 ASM 实例不仅管理一个数据库的 ASM 文件，而且可以管理多个数据库的 ASM 磁盘文件。

2.2　自动存储管理的优点

使用 ASM 管理自动存储管理减少了 DBA 维护数据库文件的工作量，提高了数据文件的可靠性和高性能的磁盘 I/O，避免磁盘热点等优点，下面给出 ASM 磁盘管理的几个优势：

① 自动磁盘管理：增加和删除磁盘的指令十分简单，而磁盘文件的重部署由 ASM 自动完成。不需要 DBA 的干预。

② 文件级镜像：通过在创建磁盘组时提供故障组，实现了文件级的冗余备份。提高了数据文件的可用性。

③ 避免磁盘热点：ASM 自动将一个文件的存储均衡在组中的磁盘上分布，提供条带化技术实现这种文件均衡分布。

④ 方便的管理数据库文件：添加和删除文件只需要“告诉”磁盘组这种操作即可，具体对文件的操作、空间分配等都由 ASM 去管理。

⑤ ASM 使用方便：其指令是 SQL 语句，对磁盘的操作都可以通过熟悉的 SQL 语句实现。

⑥ ASM 部署简单，相比第三方存储管理软件而言其免费提供，所以为中小型企业节约了成本。

2.3 ASM系统架构

这里，我们分析一下 ASM 的组成，ASM 由三部分组成：ASM 实例、磁盘组以及磁盘文件，如图 2-3 所示。

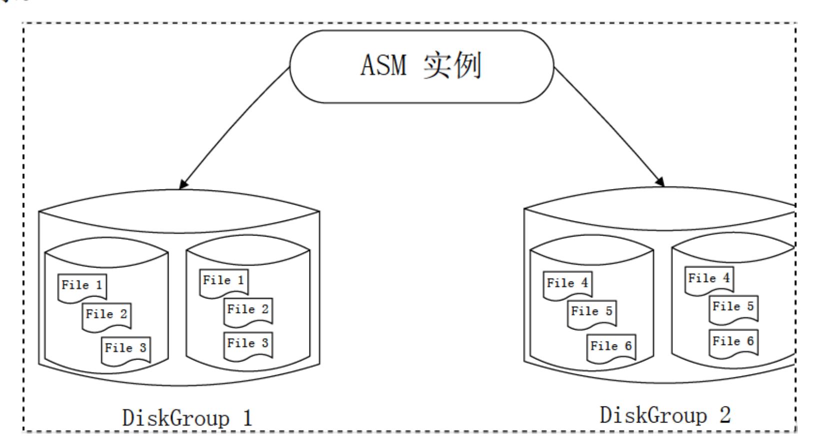


图 2-3　ASM 组成结构图

1. ASM 实例

Oracle 的 ASM 实例和普通的数据库服务器实例不同，该实例没有自己的文件系统，它只会启动相关的后台进程，挂接它所管理的磁盘组。在 Oracle 数据库中一个 ASM 实例可以管理多个数据库的数据库文件，所以在 ASM 和 Oracle 数据库之间二者是 1:N 的关系。显然，由于 Oracle 数据库的文件被 ASM 实例管理，所以要启动数据库服务器首先要启动 ASM 实例，如果要关闭数据库服务器也必须首先关闭 ASM 实例。

2. ASM 磁盘组

ASM 磁盘组是一组磁盘或磁盘分区的逻辑组合，对用户而言看到的就是一个磁盘组，将文件存储入磁盘组，其他的管理工作就由 ASM 去完成，如平衡文件的磁盘分布等操作，这些操作对用户是透明的，用户可以方便的向 ASM 中添加磁盘组，以及向磁盘组添加或删除磁盘文件，以增大磁盘的可用空间或删除不必要的磁盘（如该磁盘损坏）。

3. ASM 文件

ASM 文件存储在磁盘组上，它是 ASM 磁盘组的一部分，一个 ASM 文件必须存放在一个磁盘组中，且在该磁盘组的多个磁盘上均匀分布，即该文件在磁盘组的每个磁盘上具有条带化存储。如果磁盘组具有故障组设置，则数据库文件在每个故障组中都有完整的冗余备份，磁盘组的故障组实现了数据库文件的冗余备份，提高了数据库文件的可用性，而条带化技术实现了 I/O 的平衡，避免了某个磁盘被频繁访问，使得 I/O 活动“过热”。

2.4　ASM和CSS集群同步服务

我们已经知道 ASM 实例可以管理多个 Oracle 数据库文件，显然需要 ASM 实例与 Oracle 数据库之间进行通信，CSS 就是实现了 ASM 实例与 Oracle 数据库实例之间的同步。CSS 是集群同步的意思，在 RAC 环境下 CSS 同步多个节点的数据库实例来管理同一份数据库文件。

但是，CSS 不是一个额外需要安装的 Oracle 数据库系统软件组件，在安装 Oracle 数据库软件时 CSS 会自动安装。下面我们演示如何查看当前系统上是否运行了 CSS 服务，如图 2-4 所示。

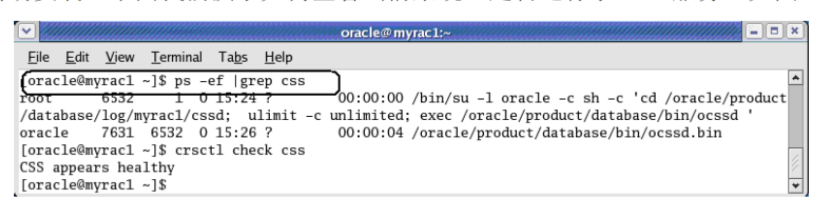


图 2-4　查看 CSS 服务状态

在图 2-4 中，使用了 ps –ef | grep css 操作系统指令查看 CSS 服务，输出说明 CSS 已经成功启动了，这里的 OCSSD 进程就提供 CSS 集群同步服务。如果 CSS 服务没有启动，即 OCSSD 进程没有运行，则可以使用 localconfig add 指令来启动 CSS 服务，如图 2-5 所示。

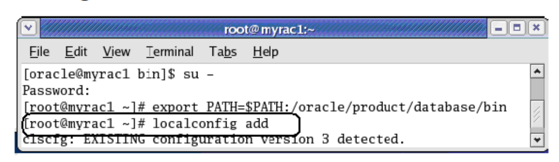


图 2-5　启动 CSS 后台进程



需要事先说明 localconfig 指令的位置，如上图我们使用 export 指令将 localconfig 指令添加到当前操作系统的默认指令搜索路径，这样在输入 localconfig add 指令时，操作系统就可以成功启动 CSS 服务。

如果在 CSS 服务启动后，由于未知的原因造成 OCSSD 进程运行异常，可以使用指令 localconfig reset 来重新启动（reset）CSS 服务，如图 2-6 所示。

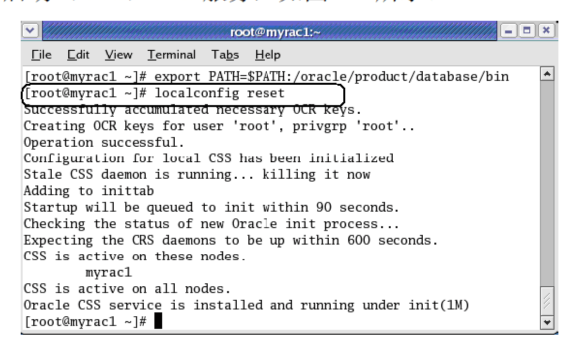


图 2-6　重启 CSS 后台进程



在图 2-6 中 localconfig 脚本会首先杀死 CSS 服务进程，然后再启动它，其实在 Oracle 中有很多类似的操作，如强制关闭数据库的指令 shutdown force，该指令会先强制关闭数据库然后再启动数据库，这里强调指令的背后实现过程。

2.5 创建ASM实例

在介绍了 ASM 自动存储管理的优点、系统架构以及与 ASM 密切相关的 CSS 服务后，本节我们着手创建一个 ASM 实例。

如果在单实例的系统上使用 DBCA 来创建 ASM 实例，则需要事先启动 CSS 服务。如果是在 RAC 集群环境下使用 DBCA 来创建 ASM 实例，则不需要启动 CSS 服务，因为在安装完 Clusterware 后执行 root .sh 脚本时会启动 CSS 集群同步服务。

下面我们演示如何在单实例环境下创建 ASM 实例。我们采用两种方式，一种是采用 DBCA 工具，一种是手工创建，这样读者就更容易理解 ASM 实例的创建过程。

无论通过 DBCA 来创建 ASM 实例还是通过手工方式创建 ASM 实例都需要启动 CSS 集群同步服务。启动方式是执行指令$ORACLE_HOME/bin/localconfig add，读者可以参考 2.4 节的内容。

1. 通过 DBCA 来创建 ASM 实例

我们不重点介绍通过 DBCA 来创建 ASM 实例，读者只要按照步骤提示就可以轻易地创建一个 ASM 实例。我们这里只给出几个关键提示。

安装完数据库管理软件后，然后通过 DBCA 来配置 ASM，这个过程会提示需要先启动 CSS，如图 2-7 所示。

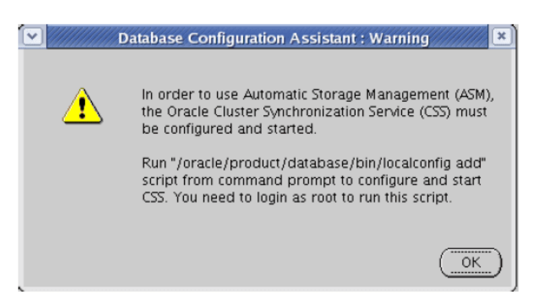


图 2-7 提示启动 CSS

从提示可以知道，为了使用 ASM 必须先启动集群同步服务（CSS），并且提示运行一个指令脚本 localconfig add。

这时，就需要使用 root 用户执行/oracle/product/database/bin/localconfig add 指令，一旦启动 CSS 后，就可以继续创建 ASM 实例了，在使用 DBCA 创建 ASM 实例的过程中会有这样一个提示，如图 2-8 所示。

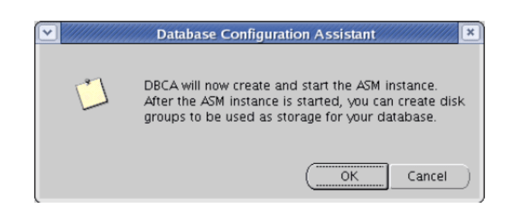


图 2-8　提示开始创建 ASM 实例

只有用户单击“OK”按钮就会创建 ASM 实例。在单实例环境下该 ASM 实例的 SID 为+ASM。如果在多实例环境下 ASM 实例的 SID 为+ASMnode#。在创建完 ASM 实例后，会提示创建 ASM 磁盘组，显然使用 ASM 存储时必须要创建 ASM 磁盘组，所以需要预先创建裸设备或者通过 ASMLib 创建 ASM 磁盘，增加 ASM 磁盘组的操作读者可以参考 2.9.2 节。下面介绍如何通过手工创建 ASM 实例，我们会给出详细步骤。

2. 手工创建 ASM 实例

先回忆一下上节讲的 ASM 实例的架构，我们知道 ASM 实例就是由多个后台进程组成的，除了传统的 SMON、PMON、CKPT 进程外，还包括特有的 RABL 等进程。与 RDBMS 实例相同，ASM 实例也需要参数文件。下面我们给出创建 ASM 实例的具体步骤。

01 安装数据库管理软件，先运行 runInstaller。此时在弹出的对话框中选择“高级选项”。单击“下一步”，然后再弹出的对话框中选中“Install database software only”单选按钮，此时我们仅安装数据库软件，如图 2-9 所示。

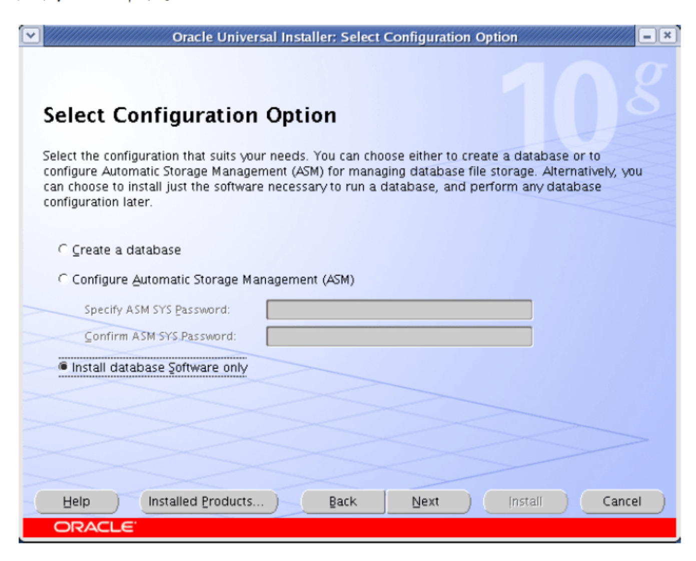


图 2-9　安装数据库管理软件

02 启动 CSS 集群同步服务。要启动 ASM 实例必须先启动 CSS 同步服务，在 Linux 系统下，我们使用 localconfig add 指令来启动该服务，启动过程如图 2-10 所示。

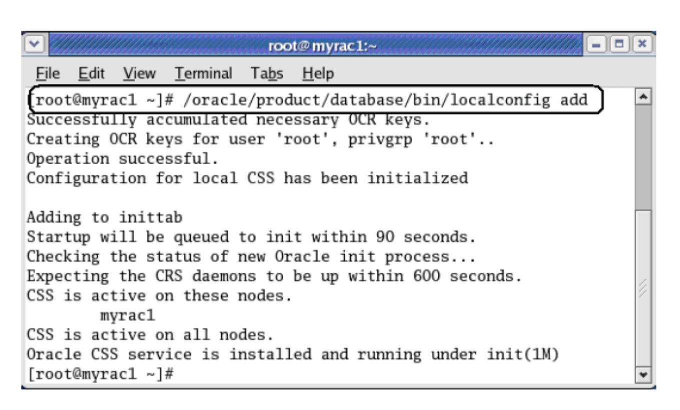


图 2-10 启动 CSS 集群服务

一旦启动后我们就可以通过 ps –ef |grep css 系统指令查看是否成功启动 CSS 服务。

03 通过 ASMLib 创建 ASM 磁盘 VOL1、VOL2、VOL3 和 VOL4。这里是创建逻辑卷，它对应操作系统的某个磁盘分区，如下例所示。

例子 2-1 创建 ASM 逻辑卷。

```
[root@myrac1~]/etc/init.d/oracleasm createdisk  VOL1 /dev/sdb3
Marking disk "VOL1 " as an ASM disk:                 [ OK ]
```

上例我们成功创建了逻辑卷 VOL1，它对应于磁盘分区/dev/sdb3，我们继续创建其他三个 ASM 逻辑卷，如下例所示。

例子 2-2 创建其他逻辑卷。

```
[root@myrac1~]/etc/init.d/oracleasm createdisk  VOL2 /dev/sdb4
Marking disk "VOL2 " as an ASM disk:                 [ OK ]
[root@myrac1~]/etc/init.d/oracleasm createdisk  VOL3 /dev/sdc1
Marking disk "VOL3 " as an ASM disk:                 [ OK ]
[root@myrac1~]/etc/init.d/oracleasm createdisk  VOL4 /dev/sdc2
Marking disk "VOL4 " as an ASM disk:                 [ OK ]
```

一旦创建成功后，我们通过 listdisks 指令查看 ASM 可以看到的磁盘，如下例所示。

例子 2-3 查看当前的 ASM 磁盘。

```
[root@myrac1~]/etc/init.d/oracleasm listdisks
VOL1
VOL2
VOL3
VOL4
[root@myrac1~]
```

输出显示当前的 ASM 磁盘有四个，分别是 VOL1、VOL2、VOL3、VOL4。

04 创建 ASM 初始化参数文件。

要创建一个 ASM 实例需要一个 ASM 实例参数，告诉 ASM 实例诸如实例类型、磁盘组位置等信息。下面是一个 ASM 实例参数文件的内容。

```
INSTANCE_TYPE=ASM
```

```
ASM_POWER_LIMIT=3
ASM_DISKSTRING='ORCL:VOL*'
ASM_DISKGROUPS=DATA , RECOVERY,DGLOG1,DGLOG2
```

下面我们解释参数具体含义，而其他参数完全可以采用 ASM 的默认值。

- **INSTANCE_TYPE**：在 Oracle 数据库环境中有两类实例，一类是 ASM，一类是 RDBMS。显然这里是创建 ASM 实例，所以该参数的值设置为 ASM。
- **ASM_POWER_LIMIT**：在 ASM 磁盘组添加或删除磁盘时，会触发 ASM 磁盘的重构，如重新分布文件、分配空间等操作，这个过程会在系统高峰期的特殊时刻影响数据库性能，此时可以通过设置该参数和告诉 ASM 对磁盘组的重构速度，该参数范围为 1~11，值越小速度越低，意味着对系统性能影响越小，该参数的默认值为 1。
- **ASM_DISKSTRING**：该参数告诉 ASM 发现磁盘时寻找的位置，即在哪里发现 ASM 磁盘或者裸设备。我们可以选择使用裸设备，也可以选择使用 ASMLib 创建的 ASM 磁盘。
- **ASM_DISKGROUPS**：说明 ASM 实例启动时要挂接的磁盘组的名称，如果实例已经启动，但是没有创建 ASM 磁盘组则会报错，提示没有挂接任何磁盘组。

我们将上述参数文件的内容保存在/home/oracle/initasm+.ora 文件中，创建过程如下所示。

例子 2-4 编辑 ASM 实例的参数文件。

```
[oracle@myrac1~] vi initasm+.ora
[oracle@myrac1~] cat initasm+.ora
INSTANCE_TYPE=ASM
ASM_POWER_LIMIT=3
ASM_DISKSTRING='ORCL:VOL*'
ASM_DISKGROUP=DATA , RECOVERY,DGLOG1,DGLOG2
[oracle@myrac1~] ls
Desktop  initasm+.ora
```

在参数文件中，我们设置了实例类型 INSTANCE_TYPE=ASM，告诉 Oracle 这是一个 ASM 实例。参数 ASM_POWER_LIMIT=3 说明 ASM 磁盘重构的速度以及磁盘组名称等信息。

05 启动 ASM 实例。

要启动 ASM 实例，首先需要 EXPORT 当前的 ORACLE_SID，然后通过 SQL*PLUS 工具启动实例，启动过程如下例所示。

例子 2-5 通过 pfile 参数文件启动 ASM 实例。

```
[oracle@myrac1~]export ORACLE_SID=+ASM
[oracle@myrac1~]sqlplus /nolog

SQL*PLUS: Release 10.2.0.1.0 - Production on Web Apr 10:15:50 2010-4-14

SQL> conn /as sysdba
Connected to an idle instance.
SQL> startup pfile=initasm+.ora
ASM instance started
Total System Global Area    79691776 bytes
Fixed Size                 1217812 bytes
Variable Size               53308140 bytes
```

```
ASM Cache                  25165824 bytes
ORA-15032: not all alterations performed
ORA-15063: ASM discovered an insufficient number of disks for diskgroup
"RECOVERY"
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DATA"
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DGLOG1"
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DGLOG2"
```



由于我们还没有创建 ASM 磁盘组，所以启动实例时会报错，先把这个问题放一下，等接下来创建完初始化参数中指定的磁盘组后，ASM 实例就能自动挂接这个磁盘组。

06 创建 ASM 磁盘组，如下例所示。

例子 2-6 创建 ASM 磁盘组。

```
SQL> create diskgrop DATA external redundancy
    disk 'ORCL:VOL1'

Diskgroup created.
```

在成功创建了 ASM 磁盘组 DATA 后，我们继续创建第二个磁盘组 RECOVERY，如下面例子所示。

例子 2-7 创建第二个 ASM 磁盘组 RECOVERY。

```
SQL> create diskgrop RECOVERY external redundancy
    disk 'ORCL:VOL2';

Diskgroup created.
```

例子 2-8 创建第三个磁盘组 DGLOG1。

```
SQL> create diskgrop RECOVERY external redundancy
    Disk 'ORCL:VOL3';

Diskgroup created.
```

例子 2-9 创建第四个磁盘组 DGLOG2。

```
SQL> create diskgrop RECOVERY external redundancy
    disk 'ORCL:VOL4';

Diskgroup created.
```

其实，此时一旦磁盘组创建成功 ASM 会自动挂接该磁盘组，下面我们关闭 ASM 实例可以看出这一点。

例子 2-10 关闭 ASM 实例。

```
SQL>shutdown immediate
ASM diskgroups dismounted
ASM instance shutdown
```

显然在关闭 ASM 实例时，提示 ASM 磁盘组已经卸载（dismounted）了，说明之前在创建完

磁盘后，ASM 已经自动挂接了初始化参数 initasm+.ora 中指定的磁盘组。

07 重新启动 ASM 实例。

我们使用 pfile 初始化参数文件重新启动 ASM 实例，如下例所示。

例子 2-11 启动 ASM 实例。

```
SQL> startup pfile=initasm+.ora
ASM instance started
Total System Global Area    79691776 bytes
Fixed Size                 1217812 bytes
Variable Size               53308140 bytes
ASM Cache               25165824 bytes
ASM diskgroups mounted
```

因为，我们在第 6 步已经成功创建了参数文件 pfile 中指定的磁盘组，所以此时启动 ASM 实例时，ASM 会自动挂接这些磁盘组。

此时，我们使用数据字典视图 v$asm_diskgroup 来查看当前磁盘组的状态，并且可以清楚地知道该磁盘组的类型，如图 2-11 所示。

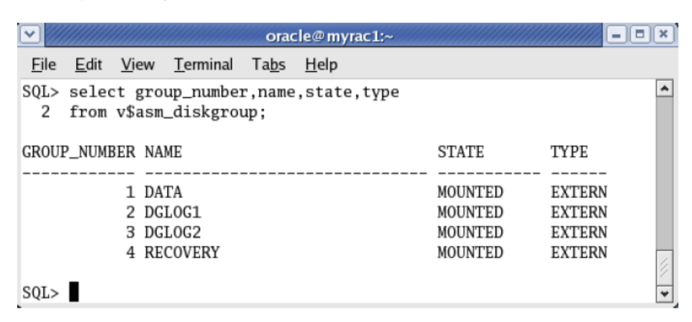


图 2-11　查看当前 ASM 磁盘组的状态

在图 2-11 中有四个磁盘组，它们分别是 DATA 、DGLOG1 、DGLOG2 和 RECOVERY，这也是我们在初始化参数文件 pfile 中设置的磁盘组，从输出看出这四个磁盘都已经挂接成功，因为其状态 STATE 都为 MOUNTED。

08 创建 spfile 文件。

我们启动 ASM 实例时使用了 pfile 文件，显然 pfile 参数文件不能动态维护，如果我们没有创建 spfile 文件，在尝试使用 startup 启动实例时，会报如下例所示的错误。

例子 2-12 无 spfile 参数文件时启动 ASM 实例。

```
SQL> startup
ORA-01078:  failure in processing system parameters
LRM-00109:   could  not  open  parameter  file  '/oracle/product/database/
dbs/init.ASM.ora'
SQL>
```

Oracle 首先会默认到目录/oracle/product/database/dbs/中搜索 ASM 参数文件，但是我们并没有创建该参数文件。下面我们创建 spfile 参数文件，如下例所示。

例子 2-13 创建 SPFILE 参数文件。

```
SQL> host cp /home/oracle/initasm+.ora /oracle/product/database/ dbs/init+ASM.ora;
SQL> create spfile from pfile;
File created.
SQL>
```



此时，已经成功创建了 spfile，上面我们创建 spfile 时，并没有给出 pfile 的具体位置，因为此时 Oracle 会默认到目录/oracle/product/database/dbs 下去搜索。我们使用 cp 指令将编辑好的参数文件拷贝到默认目录，再使用 create spfile from pfile 成功创建了动态初始化参数文件。读者也要牢记创建动态初始化参数文件，否则后期维护会比较麻烦。

下面再次使用 startup 指令启动 ASM 实例，如图 2-12 所示。

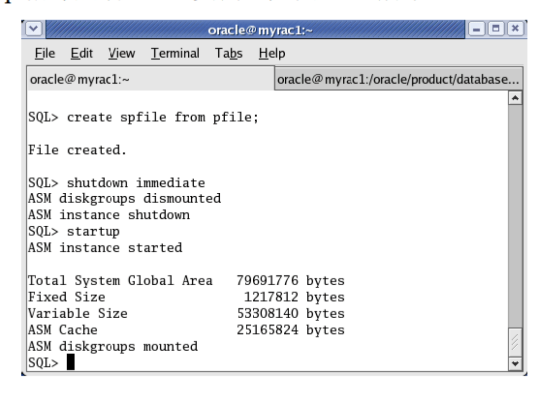


图 2-12 通过 spfile 启动 ASM 实例

上图中我们首先通过指令 create spfile from pfile 创建动态初始化参数文件，然后直接使用 startup 指令启动 ASM 实例。从输出知道 ASM 实例成功启动，所有的逻辑磁盘成功挂接。

2.6 关闭和启动ASM实例

启动 ASM 管理的 ORACLE 数据库，首先需要启动 ASM 实例。下面演示这个过程以及启动方法。

1. 启动 ASM 实例

因为在当前的数据库环境下有两个实例需要启动，在启动实例前必须告诉 Oracle 要启动哪个实例，先使用指令 export ORACLE_HOME=+ASM 告诉 Oracle 启动 ASM 实例，如图 2-13 所示。

从图 2-13 中可以清晰的看出，ASM 实例的启动其实就是分配了一块 ASM 缓存区，并挂接 ASM 管理的磁盘组。

之后可以使用 v$instance 视图查看当前实例名，如图 2-14 所示。

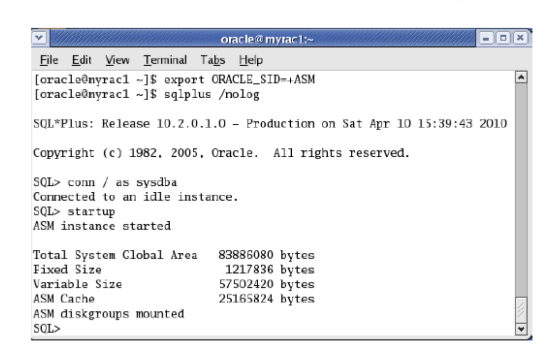


图 2-13　启动 ASM 实例

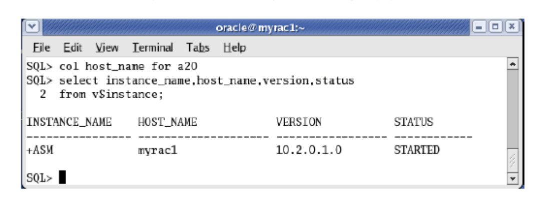


图 2-14　查询当前 ASM 实例

通过对视图 v$instance 的查询，可以清楚的看到 ASM 实例已经打开，因为 STATUS 栏状态为 STARTED。

接下来，我们查看与 ASM 实例相关的进程，如图 2-15 所示。

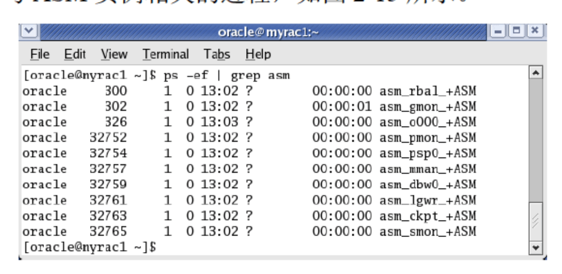


图 2-15　启动 ASM 后的 ASM 相关进程

从输出看出，所有与 ASM 实例相关的进程都已经成功启动。

2．启动数据库实例

在启动使用 ASM 作为存储管理机制的 Oracle 数据库系统之前，必须先启动 ASM 实例，这样 ASM 的磁盘组才可以成功挂接上。在启动 Oracle 数据库时，所需要的各种数据库文件才可以访问，否则根本无法启动 Oracle 数据库，提示错误如图 2-16 所示。

图 2-16 在启动数据库时，提示错误原因是无法定位参数文件，错误原因是 ORA-15077，因为 ASM 实例没有启动。所以必须先启动 ASM 实例，至于如何启动 ASM 实例读者可以参考图 2-13。

在成功启动 ASM 实例之后，我们就可以通过如图 2-17 所示的方法启动 Oracle 数据库了。

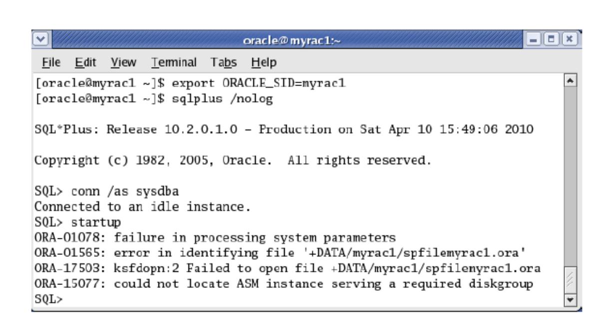


图 2-16　ASM 实例启动前启动数据库报错信息

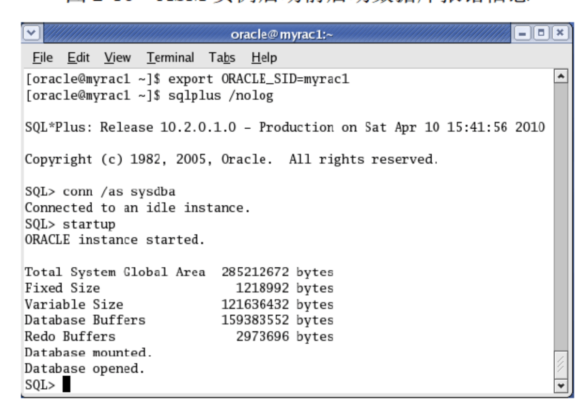


图 2-17　启动 ASM 管理的 ORACLE 数据库

一旦启动 Oracle 数据库实例之后，我们通过 v$instance 视图查看该实例的状态信息，如图 2-18 所示。

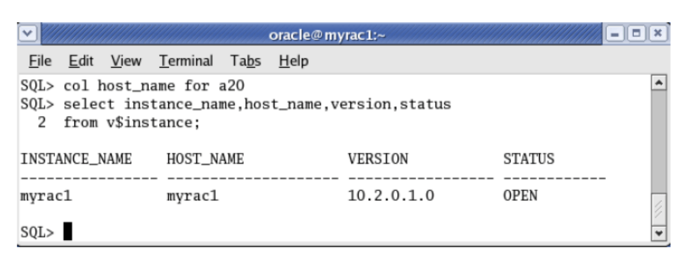
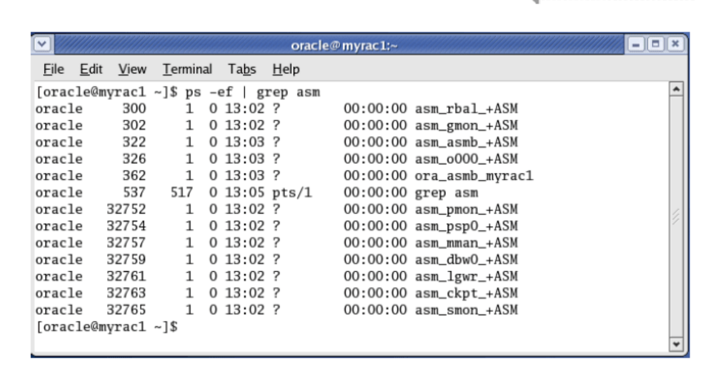


图 2-18　查看当前 Oracle 数据库实例状态

通过对视图 v$instance 的查询，可以清楚的看到数据库 myrac1 已经打开，因为 STATUS 栏状态为 OPEN。

接着我们查看采用 ASM 存储管理的 Oracle 数据库实例启动后的相关后台进程信息，如图 2-19 所示。



```
[oracle@myrac1 ~]$ ps -ef | grep asm
oracle     300     1  0 13:02 ?        00:00:00 asm_rbal_+ASM
oracle     302     1  0 13:02 ?        00:00:00 asm_gmon_+ASM
oracle     322     1  0 13:03 ?        00:00:00 asm_asmb_+ASM
oracle     326     1  0 13:03 ?        00:00:00 asm_o000_+ASM
oracle     362     1  0 13:03 ?        00:00:00 ora_asmb_myrac1
oracle     537   517  0 13:05 pts/1    00:00:00 grep asm
oracle   32752     1  0 13:02 ?        00:00:00 asm_pmon_+ASM
oracle   32754     1  0 13:02 ?        00:00:00 asm_psp0_+ASM
oracle   32757     1  0 13:02 ?        00:00:00 asm_mman_+ASM
oracle   32759     1  0 13:02 ?        00:00:00 asm_dbw0_+ASM
oracle   32761     1  0 13:02 ?        00:00:00 asm_lgwr_+ASM
oracle   32763     1  0 13:02 ?        00:00:00 asm_ckpt_+ASM
oracle   32765     1  0 13:02 ?        00:00:00 asm_smon_+ASM
[oracle@myrac1 ~]$
```

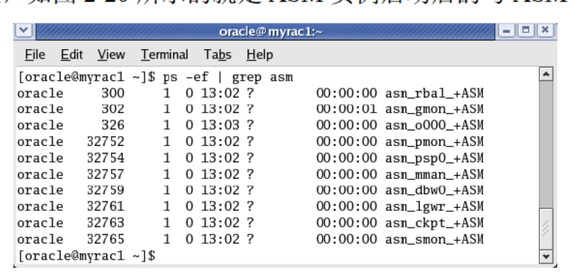
图 2-19　启动 RDBMS 后的 ASM 相关进程



从上图可以看出，除了和 ASM 自身相关的管理进程之外，数据库实例启动后多了两个进程 ora_asmb_myrac1 和 asm_asmb_+ASM。这两个进程说明 ASM 管理的 Oracle 数据库实例 myrac1 已经成功启动。

2.7　理解ASM实例架构

我们知道 ASM 实例是没有任何数据库文件的，它只包含 ASM 后台进程，所以 ASM 实例架构就包含这些后台进程，如图 2-20 所示的就是 ASM 实例启动后的与 ASM 相关的进程。



```
[oracle@myrac1 ~]$ ps -ef | grep asm
oracle     300     1  0 13:02 ?        00:00:00 asm_rbal_+ASM
oracle     302     1  0 13:02 ?        00:00:01 asm_gmon_+ASM
oracle     326     1  0 13:03 ?        00:00:00 asm_o000_+ASM
oracle   32752     1  0 13:02 ?        00:00:00 asm_pmon_+ASM
oracle   32754     1  0 13:02 ?        00:00:00 asm_psp0_+ASM
oracle   32757     1  0 13:02 ?        00:00:00 asm_mman_+ASM
oracle   32759     1  0 13:02 ?        00:00:00 asm_dbw0_+ASM
oracle   32761     1  0 13:02 ?        00:00:00 asm_lgwr_+ASM
oracle   32763     1  0 13:02 ?        00:00:00 asm_ckpt_+ASM
oracle   32765     1  0 13:02 ?        00:00:00 asm_smon_+ASM
[oracle@myrac1 ~]$
```

图 2-20　启动 ASM 后的 ASM 相关进程

这些后台进程包括 DBWR、LGWR、CKPT、SMON 等常规的 Oracle 后台进程。由于 ASM 对磁盘管理的特殊需要，ASM 有两个特殊的后台进程，他们是 RBAL（ReBALance，重平衡进程）和 ARBn（AsmReBalance，ASM 重平衡进程），其中 RBAL 管理磁盘组中的磁盘操作如打开磁盘等，而 ARBn 完成磁盘删除或添加之后的重平衡功能。

一旦使用 ASM 存储的数据库实例打开后，还会启动一个进程 ASMB，该进程和 ASM 实例以及数据库实例相连。当数据库文件变化时告诉 ASM 实例数据库文件的变化（如删除或增加），这样使得 ASM 实例完成磁盘的重平衡。也可以说 ASMB 进程是 ASM 实例与 ORACLE 数据库实例之间的通信进程，如图 2-21 所示就是启动数据库实例之后的一个和 ASMB 相关的进程。

从中我们可以看出进程 asm_asmb_+ASM 和 ora_asmb_myrac1 已经启动，这两个进程就是连接到 ASM 实例和数据库实例的 ASMB 进程。

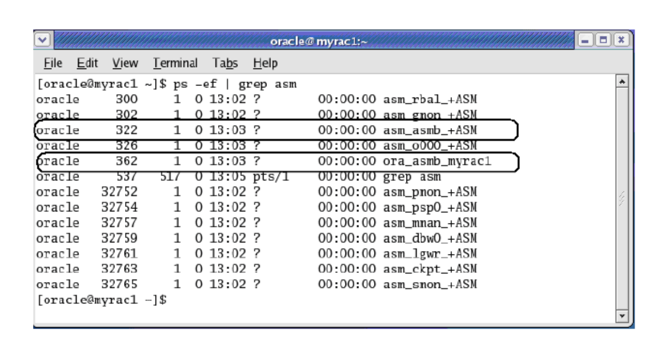


图 2-21　启动 RDBMS 后的 ASM 相关进程

2.8　ASM命令行管理工具

管理 ASM 的文件系统，如可以查看 ASM 管理了哪些文件、创建或删除目录、检查磁盘空间等操作。

ASMCMD 工具可以实现管理 ASM 磁盘，但是运行 ASMCMD 工具需要几个条件，如下所示。

① 使用 SYSDBA 权限登录 ASM 实例，启动 ASM 实例并挂接磁盘组。

② 设置 ORACLE_SID 和 ORACLE_HOME 环境变量，指定 ORACLE_SID 为 ASM 实例名。如果是单实例环境 ORACLE_SID 默认为+ASM，如果是 RAC 环境 ORACLE_SID 值为+ASMnode_number。

启动 ASM 实例的具体操作如下例所示。

例子 2-14 启动 ASM 实例。

```
[oracle@myrac1~]$ export ORACLE_SID=+ASM     //说明要启动的实例类型
[oracle@myrac1~]$ sqlplus /nolog             //启动 SQL*PLUS 工具

SQL*PLUS: Release 10.2.0.1.0 - Production on Mon Apr 12 12:23:01 2010-4-12
Copyright (c) 1982, 2005, Oracle. All rights reserved.

SQL> conn /as sysdba
Connected to an idle instance.
SQL> startup                                 //启动 ASM 实例
ASM instance started

Total System Global Area 83886080 bytes
Fixed Size              1217836 bytes
Variable Size            57502420 bytes
    ASM Cache            25165824 bytes
ASM diskgroups mounted
```

在 ASM 实例启动并成功挂接磁盘组后，退出 SQLPLUS 工具，回到系统，运行 ASMCMD 指令，并查询 ASMCMD 支持的指令（ASMCMD>help），如图 2-22 所示。

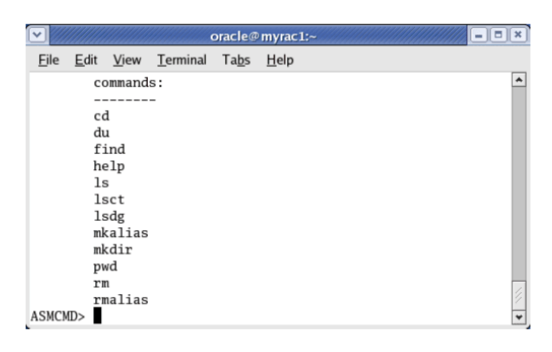


图 2-22　asmcmd 管理指令

下面我们介绍相关的几个操作，并进行实例演示。

1. ASM 目录

ASM 目录具有层次结构，在创建 ASM 文件时，系统会创建默认的文件目录结构，该结构的文件名也称为完全格式的文件名，以符号+为根目录，使用 ls 指令显示当前目录文件内容，使用 cd 指令切换目录，使用 pwd 指令查看当前路径，如下例所示。

例子 2-15 使用 pwd 指令查看当前路径。

```
[oracle@myrac1~]$ export ORACLE SID=+ASM
[oracle@myrac1~]$asmcmd
ASMCMD>pwd
+
ASMCMD>
DATA/
DGLOG1/
DGLOG2
RECOVERY/
ASMCMD>
```

接着，我们使用 cd 指令切换目录，并查看当前路径，如下例所示。

例子 2-16 使用 cd 指令切换目录，并查看当前路径。

```
ASMCMD>cd data/myrac1/datafile
ASMCMD>pwd
+data/myrac1/datafile
ASMCMD>ls
MYTBS1.263.716420541
SYSAUX.257.716341253
SYSTEM.256.716341203
UNDOTBS1.258.716341285
USERS.259.716341289
ASMCMD>
```

我们可以通过 ASMCMD mkdir 命令创建用户目录作为系统生成目录的子目录，如下例所示。

例子 2-17 通过 ASMCMD mkdir 命令创建用户目录。

```
ASMCMD> mkdir +dgroup1/mydir
```

```
ASMCMD>cd +data/mydir
ASMCMD>pwd
+data/mydir
ASMCMD>
```



ASM 不会在用户目录中放置系统文件，单用户的别名可以放置在该目录下，以便于用户维护。

2. 别名（Alias）

别名是指向系统生成的文件名的一个更友好的文件名字，与 Unix 系统的符号链接类似。使用 ASMCMD 的 mkalias 命令创建别名，用户可以在磁盘组路径或者任何系统生成或用户创建的子目录下创建别名，列出别名指向的链接文件。

如图 2-23 所示，我们使用 ASMCMD 创建别名。

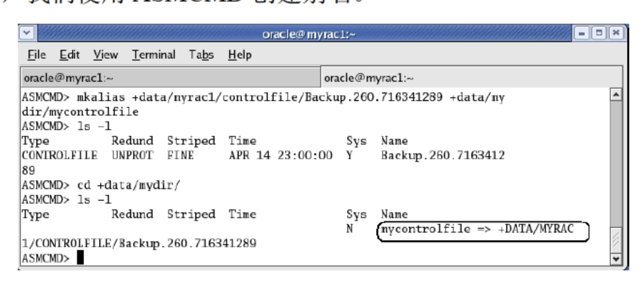


图 2-23　创建别名

在上图中，我们先为文件+data/myrac1/controlfile/Backup.260.716341298 创建别名为+data/mydir/mycontrolfile，同时我们进入目录+data/mydir。并通过指令 ls –l 来查看该目录下别名对应的链接。上图中圈起来的内容就是别名对应的链接。下面我们给出一个 ASMCMD 指令的汇总表，供读者参考，如表 2-1 所示。

表 2-1　ASMCMD 命令参考

命令	描述
cd	改变当前目录到指定的目录
du	显示指定目录及其子目录中 ASM 文件占用的磁盘空间
exit	退出 ASMCMD
find	在指定目录下列出包含指定名称（使用通配符）的路径
help	显示 ASMCMD 命令的语法和描述
ls	列出 ASM 目录的内容，指定文件的属性
lsct	列出关于当前 ASM 客户端信息，也就是使用 ASM 实例管理文件的数据库信息
lsdg	列出所有磁盘组及其属性
mkalias	为系统生成的文件创建一个别名
mkdir	创建 ASM 目录
pwd	显示当前路径
rm	删除指定 ASM 文件或路径
rmalias	删除指定别名

2.9　管理ASM磁盘组

使用 ASM 存储和管理的数据库文件都存储在磁盘组中，具体磁盘 I/O 平衡，磁盘组重构都由 ASM 自己完成。本节我们讲解 ASM 磁盘组的优势，以及这些优势背后的技术支持是什么。接着介绍 DBA 实际维护需要的磁盘组操作，像如何创建磁盘组、添加和删除磁盘等。

2.9.1　使用 ASM 磁盘组管理文件的优势

我们知道使用 ASM 自动存储管理的好处体现在“自动化”，即减轻了 DBA 管理数据库文件的负担，但是我们还需要进一步理解通过这个“自动化”背后具体做了那些工作。其实，使用 ASM 管理数据库文件的主要优势在于其内在的平衡 I/O 和数据库文件的冗余保护。也就是说通过 ASM 磁盘组存储的数据库文件不会在磁盘组中的磁盘中引起“热点”，同时对数据库文件起到冗余备份的作用，那么 ASM 是如何做到这些的呢，我们下面依次分析。

1．条带化技术

在 ASM 磁盘组中有多个磁盘，我们存储数据库文件时，ASM 会自动把数据库文件进行条带化分割，将文件平衡的存储在多个磁盘上，如图 2-24 所示。

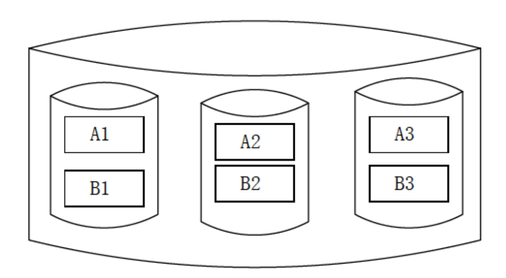


图 2-24　磁盘组实现条带化的示意图

上图中有两个数据库文件 A 和 B，通过条带化将文件分割，分别存储在磁盘组中的三个磁盘上。每个文件被分成三个“条带”，这样就平衡了磁盘的存储。ASM 要求磁盘组中的磁盘型号和容量要相同，这样就保证了存储的平衡以及管理的方便。

2．ASM 镜像技术

镜像的含义就是对数据文件的冗余备份，即如果某个存储数据文件的磁盘损坏，磁盘组会从镜像磁盘中读取该数据文件，实现对故障磁盘的重构。ASM 提供了三类镜像类型，分别是外部冗余（EXTERNAL REDUNDANCY）、常规冗余(NORMAL REDUNDANCY)和高冗余(HIGH REDUNDANCY)。

在介绍 ASM 镜像类型之前，我们需要引入一个概念——故障组。

我们知道 ASM 磁盘组由多个物理磁盘或磁盘分区组成。比如一个磁盘组有四个磁盘成员，但是这四个磁盘成员由一个磁盘控制器控制。如果这个磁盘控制器故障，就会造成磁盘组中的所有磁盘都无法访问，虽然在每个磁盘中有对数据库文件的冗余备份，但是此时的备份是无法访问，这样的一个磁盘组就成为故障组。通过故障组的设置就大大提高了 ASM 管理的数据库文件的可用性，

因为如果一个磁盘控制器损坏，其他磁盘控制器相关的磁盘组具有该数据库文件的冗余备份，如果有三个故障组就会有两份相同数据库文件的冗余备份。

图 2-25 就是一个 ASM 磁盘组带有三个故障组的示意图，其中每个故障组包括两个磁盘。

当向磁盘组 Diskgroup1 存储数据库文件时，此时在三个故障组中分别存储该文件的完整备份，实现了数据文件存储的冗余，这样就提高了数据文件的高可靠性，而在每个故障组中，ASM 实现了条带化存储，这样就实现了平衡磁盘 I/O。

在理解了故障组之后，我们分别介绍 ASM 镜像类型的含义。

（1）外部冗余（EXTERNAL REDUNDANCY）

外部冗余是没有镜像的一种冗余，显然这种冗余不是“真冗余”，即在磁盘组中没有故障组。对于文件可靠性要求不高的磁盘组可以采用这种冗余策略。

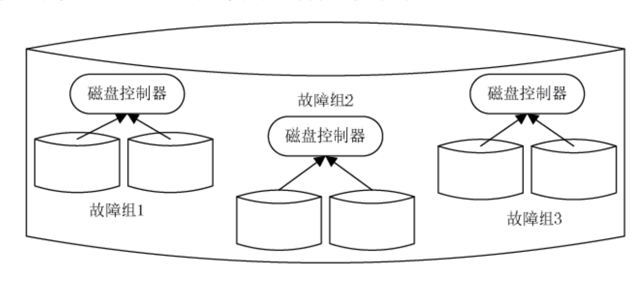


图 2-25　故障组示意图

（2）常规冗余（NORMAL REDUNDANCY）

常规冗余要求磁盘组中具有两个故障组，每个故障组一个磁盘控制器。

（3）高冗余（HIGH REDUNDANCY）

高冗余要求磁盘组中具有三个故障组，每个故障组一个磁盘控制器。

2.9.2　创建磁盘组

创建磁盘组有多种方式，一是通过 DBCA 工具来配置 ASM 增加磁盘组，一个是通过命令行方式手工创建磁盘组。下面我们演示如何通过这两种方式创建磁盘组。

首先我们要创建 ASM 磁盘，我们使用 ASMlib 方式创建，在第 1 章中已经安装并配置了 ASM，这里我们就创建两个磁盘，并在这两个磁盘基础上创建一个新的磁盘组。

首先给出使用 DBCA 工具创建的方式，前提是必须已经创建了 ASM 磁盘或者裸设备。

1．DBCA 来创建磁盘组。

启动 DBCA 在图 2-26 所示的窗体中选中“Congfigure Automatic Storage Management”单选按钮来配置 ASM 存储管理。

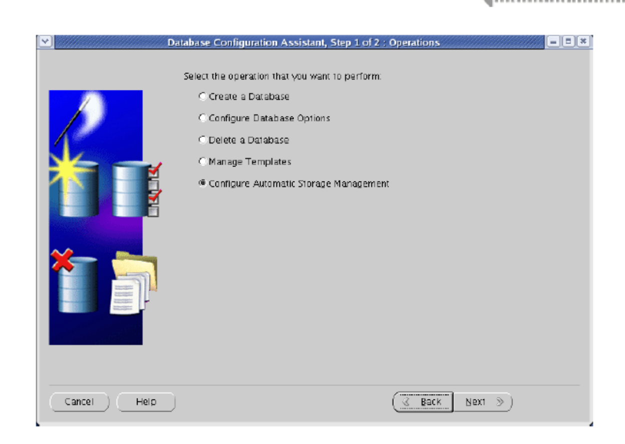


图 2-26　启动 DBCA 选择配置 ASM 选项

然后，单击图 2-26 所示的“Next”按钮，显示如图 2-27 所示的界面。

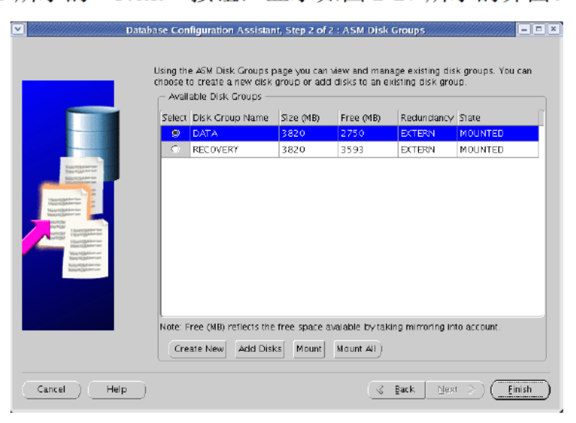


图 2-27　当前可获得的磁盘组

在图 2-27 中，我们可以创建新磁盘组、在当前已存在的磁盘组中添加磁盘，如果磁盘组没有被挂接可以手动挂接这些磁盘组。单击上图的“Create New”按钮来创建新的磁盘组，如图 2-28 所示。

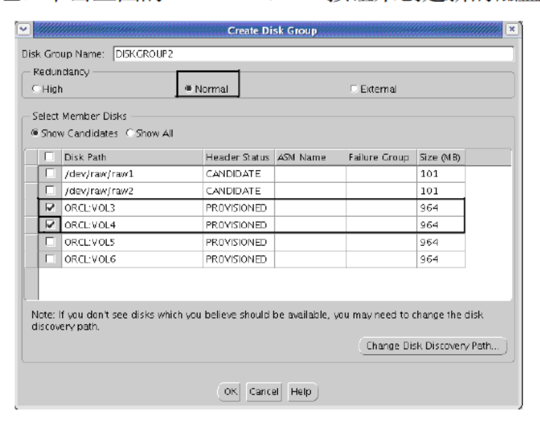


图 2-28　创建新的磁盘组 DISKGROUP2

在上图中，选择 ASM 磁盘 VOL4 和 VOL5。然后单击“OK”按钮，如图 2-29 所示。

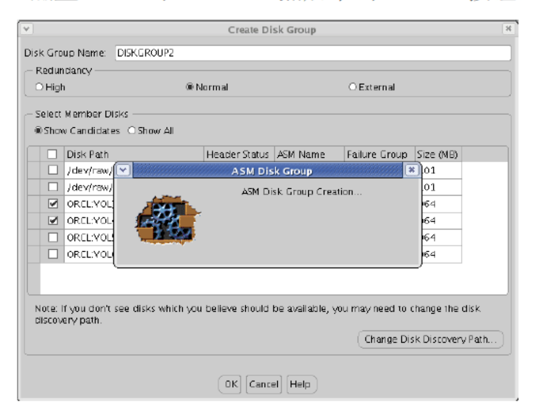


图 2-29　创建新磁盘 DISKGROUP2

当图 2-29 创建 ASM 磁盘组的过程结束后，自动返回图 2-30 所示界面，此时在磁盘组当中可以看到多了一个磁盘组 DISKGROUP2，且状态为 MOUNTED，说明该磁盘组可供使用。以后每次启动 ASM 实例时就会自动挂接该磁盘组。

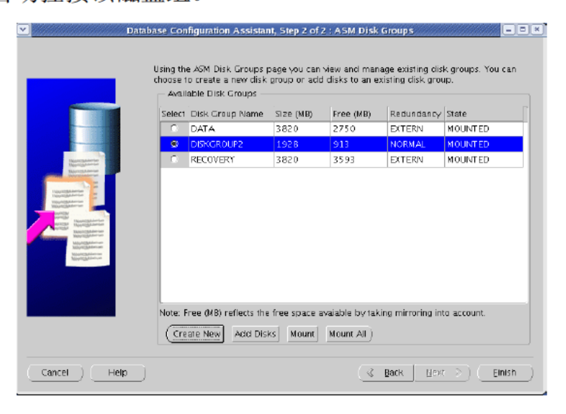


图 2-30　当前可以获得的 ASM 磁盘组



在图 2-31 中，我们看到粗黑线条所示的部分是刚刚添加的磁盘组成员信息，其中有一列为“Failure Group”，我们翻译为“故障组”。其中磁盘组成员 VOL3 对应的故障组名为 VOL3，而磁盘组成员 VOL4 对应的故障组成员为 VOL4，其实这两个磁盘组成员对应的故障组名称是 Oracle 自动加上去的，这里的关键是理解什么是故障组。

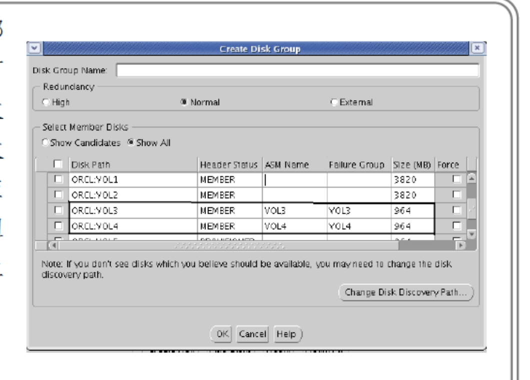


图 2-31　成功创建的磁盘组信息

2．手工创建磁盘组

在 ASM 的启动过程中，首先分配内存，读取 ASM 磁盘组初始化参数，再 MOUNT 磁盘组。如果需要添加磁盘，显然需要在 NOMOUNT 状态，即没有 MOUNT 任何磁盘组的情况下添加新磁盘组。这种情况下，ASM 实例就必须以 NOMOUNT 状态启动。

下面我们演示如何创建磁盘组，如图 2-32 所示。

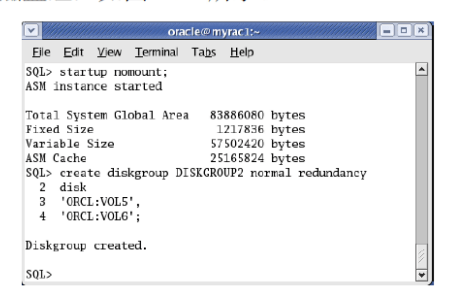


图 2-32　创建 ASM 磁盘组

上图中我们创建了一个磁盘组 DISKGROUP2，它由两个磁盘分区组成，VOL5 和 VOL6 都是 ASM 磁盘。为了验证我们创建的结果，我们不使用 DBCA 这样的图形化工具验证磁盘组 DISKGROUP2 的创建结果，而是使用数据字典视图 v$asm_disk，如图 2-33 所示。

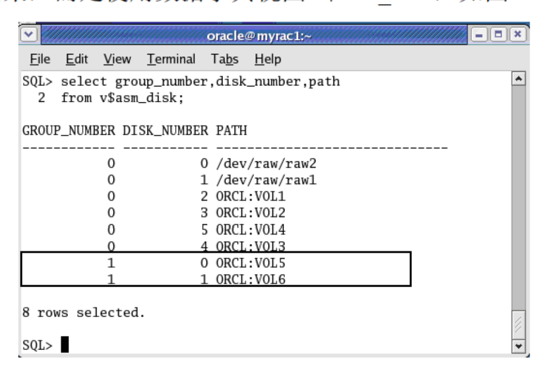


图 2-33　查询磁盘组 DISKGROUP2 的信息

从上图所示的查询可以看出，刚刚创建的磁盘组 DISKGROUP2 的 GROUP_NUMBER 为 1，该组中的两个 ASM 磁盘 VOL5 和 VOL6 的 DISK_NUMBER 分别为 0 和 1。

同时我们可以使用 asmcmd 的指令查看创建磁盘组 DISKGROUP2 的结果，如图 2-34 所示。

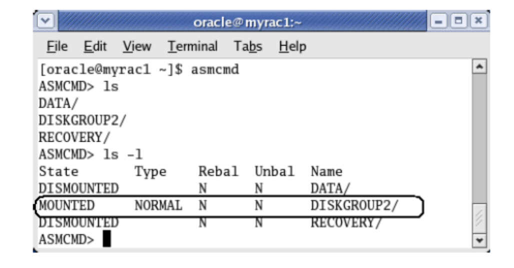


图 2-34　使用 ASMCMD 指令查看磁盘组创建结果

2.9.3 向磁盘组添加磁盘

在上节我们已经成功创建了一个磁盘组 DISKGROUP2，该磁盘组包含两个逻辑磁盘 VOL5 和 VOL6，下面我们演示如何向该磁盘组中添加磁盘，以增加该磁盘组的物理空间，增加数据库文件的高可用性。因为一旦增加了新的磁盘，ASM 会触发数据库文件的自动平衡操作，将数据文件在当前磁盘组的所有磁盘中进行动态分配，实现数据库文件的镜像化。

那么，如何向当前磁盘组中增加磁盘呢？在 Oracle 的 ASM 实例环境下使用如下指令。

```
ALTER DISKGROUP group_name ADD DISK disk_name
```

下面通过例子演示增加磁盘的方法，这里我们向磁盘组 DISKGROUP2 增加一个 ASM 磁盘 VOL4，如图 2-35 所示。

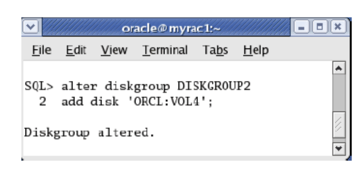


图 2-35 添加磁盘

我们再次通过数据字典视图 v$asm_disk 验证是否成功添加磁盘到 DISKGROUP2 磁盘组，如图 2-36 所示。

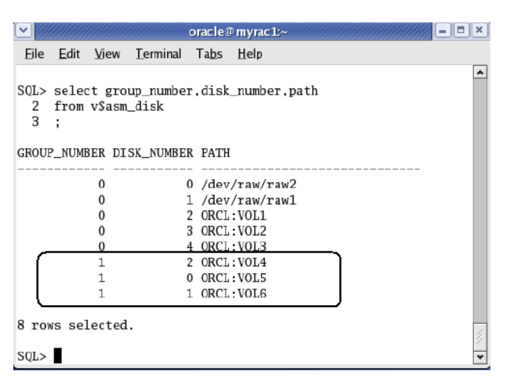


图 2-36 验证添加磁盘结果

从图 2-36 所示的输出可以看出，ASM 磁盘 VOL4 添加到磁盘组 1 中，GROUP_NUMBER 相同的组，DISK_NUMBER 按照添加顺序增加，VOL4 是最后添加的，其 DISK_NUMBER 号按顺序增加，其 DISK_NUMBER=2。在下节，我们将继续使用磁盘组 DISKGROUP2 作为例子，学习如何删除磁盘组中的磁盘以及如何删除整个磁盘。

2.9.4 删除磁盘和磁盘组

如果需要删除一个磁盘组中的磁盘，使用 alter diskgroup 指令实现，具体操作如图 2-37 所示，我们删除了磁盘组 DISKGROUP2 中的 ASM 磁盘 VOL6。

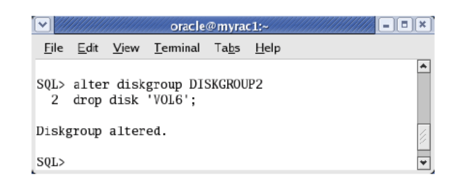


图 2-37　删除磁盘

下面我们通过数据字典视图 v$asm_disk 来验证删除结果，即在删除 ASM 磁盘 VOL6 之后，磁盘组 1 中是否还存在该磁盘，如图 2-38 所示。

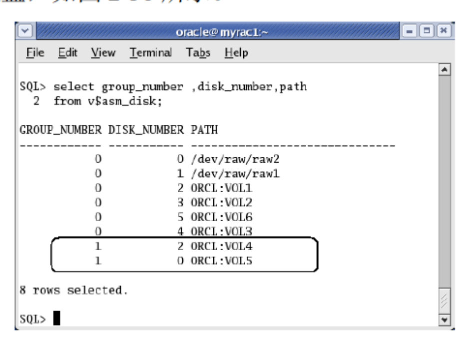


图 2-38　验证磁盘删除结果

从上图可以看出，磁盘组 DISKGROUP2 中只有两个 ASM 磁盘：ORCL:VOL4 和 ORCL:VOL5。说明已经成功删除磁盘 VOL6。

接着，我们演示如何删除磁盘组以及验证删除磁盘组的结果，如图 2-39 所示。

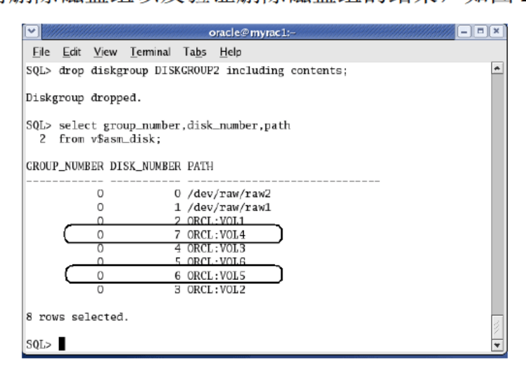


图 2-39　删除磁盘组以及验证删除结果

在磁盘组 DISKGROUP2 中有两个 ASM 磁盘 VOL4 和 VOL5，当我们使用 drop diskgroup 指令删除该磁盘组后，在 v$asm_disk 视图中已经看不到磁盘组的存在了。这说明我们已经成功删除磁盘组 DISKGROUP2。

同样可以使用 ASMCMD 指令查看是否还有 DISKGROUP2 这个磁盘组，如图 2-40 所示。

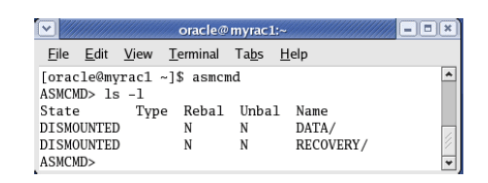
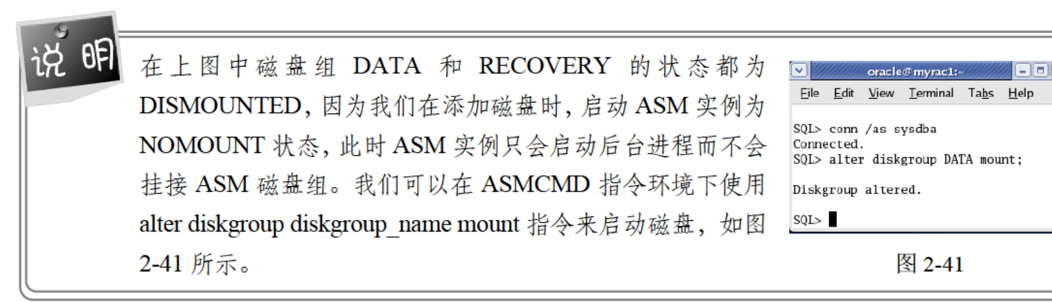


图 2-40　验证磁盘组删除结果

上图说明，当前 ASM 实例只有两个磁盘组 DATA 和 RECOVERY，所以通过 ASMCMD 指令也可以确定我们已经成功删除磁盘组 DISKGROUP2。



在上图中磁盘组 DATA 和 RECOVERY 的状态都为 DISMOUNTED，因为我们在添加磁盘时，启动 ASM 实例为 NOMOUNT 状态，此时 ASM 实例只会启动后台进程而不会挂接 ASM 磁盘组。我们可以在 ASMCMD 指令环境下使用 alter diskgroup diskgroup_name mount 指令来启动磁盘，如图 2-41 所示。

图 2-41

上图中，修改结果提示“Diskgroup altered”说明已经成功修改了磁盘组 DATA 的状态，磁盘组 DATA 已经挂接到 ASM 实例。

2.9.5　平衡磁盘组

ASM 的一个特点就是减少磁盘热点，即避免一个磁盘过多的磁盘 I/O 出现。无论是增加磁盘还是删除磁盘都会触发 ASM 的磁盘平衡功能，这个功能是通过 RABL（重平衡进程）实现的。显然 ASM 磁盘平衡的操作是一个自动的动态行为，不需要用户的参与。

但是读者可以想象磁盘的重平衡对存储在磁盘上的文件进行重新部署，这样势必会影响数据库系统的性能，可以想象如果在系统运行高峰期间发生磁盘重平衡显然会影响系统的性能，所以对于磁盘的添加和删除行为应该在系统业务清闲的时间段进行。

在 ASM 实例中有一个参数 ASM_POWER_LIMIT，该参数的作用是控制对磁盘的平衡速度，该参数值的范围 1~11，参数值越大说明磁盘平衡操作速度越快。可以根据性能需要调整该参数来影响对某个磁盘组的平衡速度。图 2-42 中显示了参数的信息，参数 ASM_POWER_LIMIT 的默认值为 1。

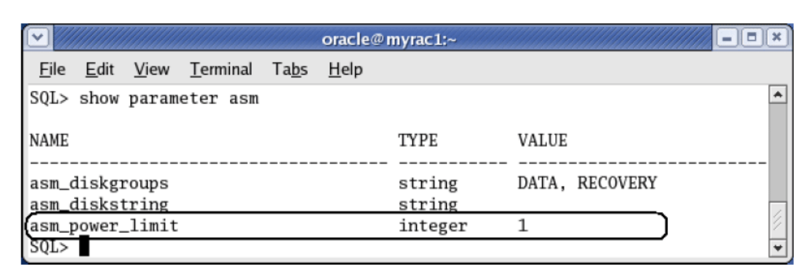


图 2-42　查询参数 ASM_POWER_LIMIT 信息

下面我们介绍如何修改参数 ASM_POWER_LIMIT 的值。我们使用 alter diskgroup 指令来修改，注意 POWER 参数的值，该值就是修改参数 ASM_POWER_LIMIT 的值。方法如下例所示。

例子 2-18 修改参数 ASM_POWER_LIMIT 的值。

```
SQL> alter diskgroup DATA add disk 'VOL3'  rebanlance power 5
     Diskgroup altered;
```

上述的 POWER 子句告诉 ASM 在增加磁盘后，执行磁盘平衡的速度为 5，该值修改为 5，显然比默认值要大，所以对于磁盘组 DATA 的重平衡速度也要比默认要快一些。

2.9.6 MOUNT 和 DISMOUNT 磁盘组

ASM 磁盘组在 ASM 实例启动时自动 MOUNT，此时 ASM 磁盘组中的数据库文件就是可访问的，当 ASM 实例关闭时 ASM 磁盘组也会自动 DISMOUNT。在 ASM 实例创建时也会 MOUNT 磁盘组，用户删除一个 ASM 磁盘组后，ASM 实例会自动 DISMOUNT 删除的磁盘组。

当 ASM 实例运行时，可以使用 ALTER DISKGROUP diskgroup_name DISMOUNT 指令来卸载挂接的 ASM 磁盘组。使用 ALTER DISKGROUP diskgroup_name MOUNT 指令来挂接磁盘组，如图 2-43 所示，我们先 DISMOUNT 磁盘组 RECOVERY。

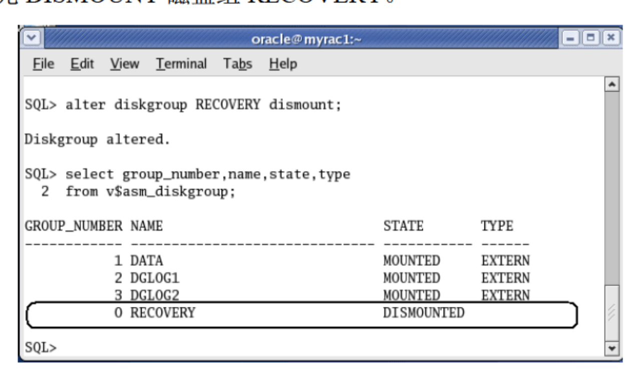


图 2-43　卸载 ASM 磁盘

从上图看出，磁盘组 RECOVERY 的状态为 DISMOUNTED，说明已经成功卸载。下面，我们使用 ALTER DISKGROUP diskgroup_name MOUNT 指令来挂接磁盘组 RECOVERY，如图 2-44 所示。

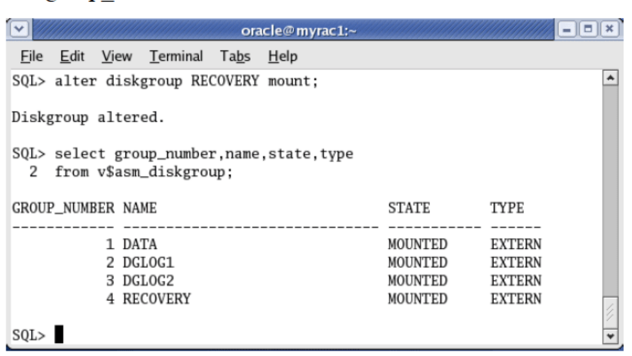


图 2-44　挂接 ASM 磁盘组

除了上述两个指令来 MOUNT 和 DISMOUNT 磁盘组外，还可以依次 MOUNT 和 DISMOUNT 所有磁盘组。如下指令所示。

例子 2-19 MOUNT 和 DISMOUNT 所有磁盘组。

```
SQL>alter diskgroup all mount;
SQL>alter diskgroup all dismount;
```



还有一种情况就是当前要 DISMOUNT 的磁盘组中包含已经打开的文件，此时就无法实现正常卸载磁盘组，如果这种情况下需要 DISMOUNT 磁盘组，需要使用 FORCE 参数，如下例所示。

例子 2-20 使用 FORCE 参数 DISMOUNT 所有磁盘组。

```
SQL>alter diskgroup all dismount force;
```

2.10 管理ASM文件

ASM 文件是一种 OMF 格式的文件，不容易记忆和使用，但是 ASM 提供了别名方式以及允许用户创建自己的目录，这大大提高了这种文件的友好性。本节我们从 ASM 磁盘组文件名结构入手，介绍文件结构组成，然后介绍创建目录以及创建别名，最后我们简要给出 ASM 文件模板的作用，并给出两个使用模板创建文件的例子，以供读者学习。

2.10.1 ASM 磁盘组文件名结构

ASM 文件名结构是系统默认产生的，系统所产生的文件名可以看出分级的目录结构，如下例所示。

```
+DATA/myrac1/datafile/SYSAUX.257.716341253
+DATA/myrac1/datafile/SYSTEM.256.716341203
+DATA/myrac1/datafile/UNDOTBS1.258.716341258
+DATA/myrac1/datafile/USERS.259.716341289
```

我们分析一下系统产生的这个分级目录结构，首先+号表示所有文件根目录，而 DATA 是所有数据库文件的父目录，myrac1 是所有 myrac1 数据库文件的父目录，而 datafile 包含所有 myrac1 数据库的数据文件。

2.10.2 ASM 磁盘组中目录管理

显然系统产生的分级文件名结构读者不易记忆，但是用户可以根据需要对文件建立别名，同时创建自己易于记忆的目录来存储这些别名。本节我们讲述如何管理用户自己的目录。

1. 创建目录

我们给出一个例子说明如何创建 ASM 磁盘子目录，如图 2-45 所示。

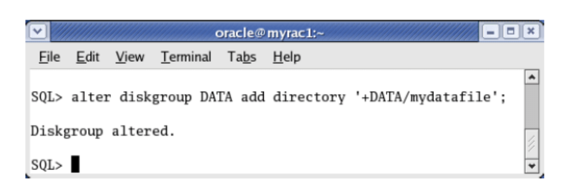


图 2-45　在磁盘组创建新目录

上图提示创建成功，我们通过 ASMCMD 指令查看该目录是否存在，如图 2-46 所示。

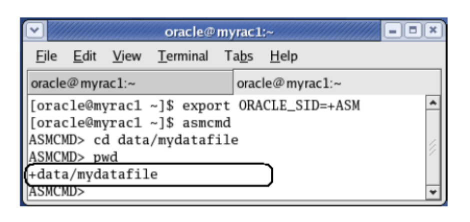


图 2-46　查询新目录是否创建成功

读者需要注意，在 ASM 磁盘组中创建新目录时，必须保证子目录存在，如下例所示。

```
SQL>alter diskgroup DATA add directory '+DGLOG1/mylogfile/log_group1.
```

如果 ASM 磁盘组 DGLOG1 中没有子目录+DGLOG1/mylogfile，则上述目录创建失败，如图 2-47 所示。

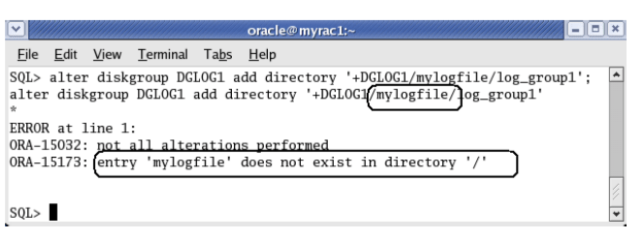


图 2-47　创建目录失败

从上图中的错误提示很容易看出，由于目录 mylogfile 不存在，所以不能在“不存在”的目录下创建新目录。

2．修改目录名

使用 ALTER DISKGROUP …RENAME … TO…. 指令修改用户创建的目录名，如图 2-48 所示，将目录“+data/mydatafile”改为“+data/mydbfile”。

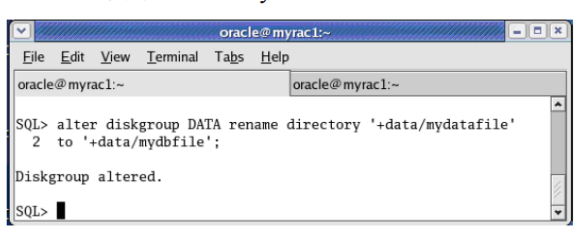


图 2-48　修改目录名

3. 删除目录

如果不需要用户自己创建的目录，可以删除它，使用指令 ALTER DISKGROUP …DROP DIRECTORY…FORCE，如图 2-49 所示。

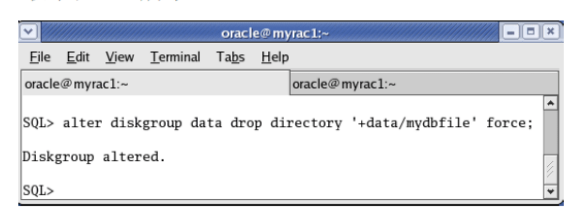


图 2-49 删除目录



在上述删除磁盘目录的指令中使用了 force 子句，该子句的作用是若要删除的目录下有子目录则一起删除。若存在子目录的情况下不使用 force 子句则无法成功删除目录。

2.10.3 添加和删除别名

在上节我们成功演示了创建+DATA/mydatafile 子目录，这样为我们创建别名创造了条件，就可以把别名存储在该目录下，方便对文件的操作。下面我们演示如何创建别名。

首先我们必须先创建一个子目录+DATA/mydatafile，然后为数据文件创建别名，如图 2-50 所示。

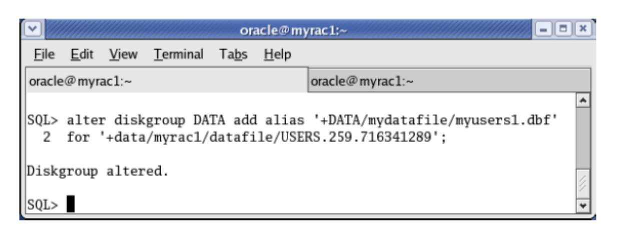


图 2-50 创建文件别名

在上例中，我们使用 ALTER DISKGROUP…ADD ALIAS …FOR 指令为表空间 users 中的数据文件 USERS.259.716341289 创建了别名+data/mydatafile/myusers1.dbf，这样数据文件就容易记忆了。在管理起来也方便，以后只需要通过别名来操作数据文件 USERS.259.716341289 了。

也可以修改别名，如图 2-51 所示。

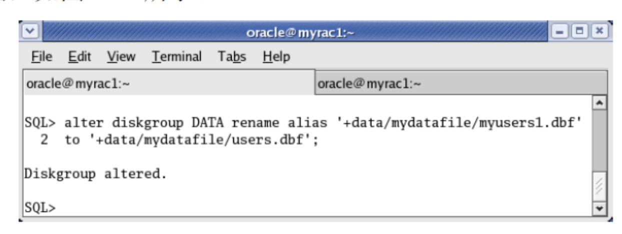


图 2-51 修改别名

下面我们通过 ASMCMD 指令来查看该别名是否存在，如图 2-52 所示。

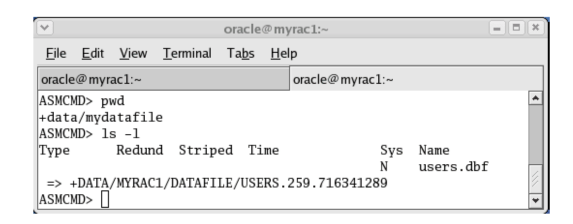


图 2-52　验证别名创建结果

在上图 2-52 中，目录+data/mydatafile 下创建了一个文件名 users.dbf，该名字指向一个绝对路径下的文件+DATA/MYRAC1/DATAFILE/USERS.259.716341258。

在不需要别名时，可以使用如下指令删除别名。

例子 2-21　删除 ASM 文件别名。

```
SQL>alter diskgroup DATA drop alias '+DATA/mydatafile/users.dbf ';
```

2.10.4　删除文件

可以通过删除别名文件的方式删除别名和别名对应的文件，也可以通过删除文件的方式删除别名。

```
SQL> alter diskgroup DATA drop file '+data/mydatafile/users.dbf'
```

上述指令将删除别名+data/mydatafile/users.dbf，并删除别名对应的文件。还可以直接删除文件，如下例所示。

例子 2-22　删除 ASM 磁盘文件。

```
SQL>alter diskgroup DATA drop file
'+DATA/MYRAC1/DATAFILE/USERS.259.716341258';
```

2.10.5　使用 ASM 文件模板

模板是与文件冗余和条带化对应起来的。显然不同的文件有不同冗余要求，并且文件的冗余和磁盘组类型也有关系，比如一个 Redo LOGFILE，需要在 ASM 实例参数 DB_CREATE_ONLINE_LOG_DEST_n 指定的磁盘组中冗余存储。

当创建指定的文件如数据文件、控制文件以及重做日志文件等时，会默认根据其特点使用其对应的模板来实现文件冗余和条带化存储。下面给出一个例子。

使用 datafile 模板创建一个表空间，默认在磁盘组 DATA 上创建表空间对应的数据文件，文件存储在+DATA/myrac1/datafile/目录下，如下例所示。

例子 2-23 使用 datafile 模板创建一个表空间。

```
SQL>create tablespace mytbs1 datafile ;

Tablespace created.
SQL>
```

然后，我们使用 ASMCMD 指令查询该表空间对应的数据文件信息，如图 2-53 所示。

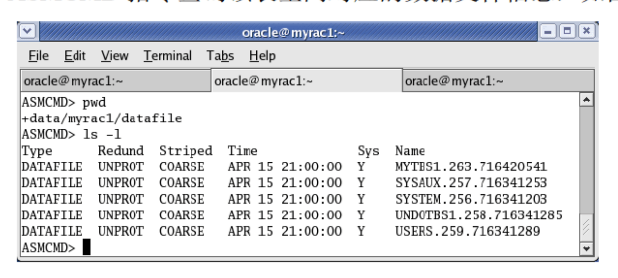


图 2-53　新建表空间 mytbs1 对应的数据文件

下面我们再使用 onlinelog 模板来创建日志组和日志文件，如图 2-54 所示。

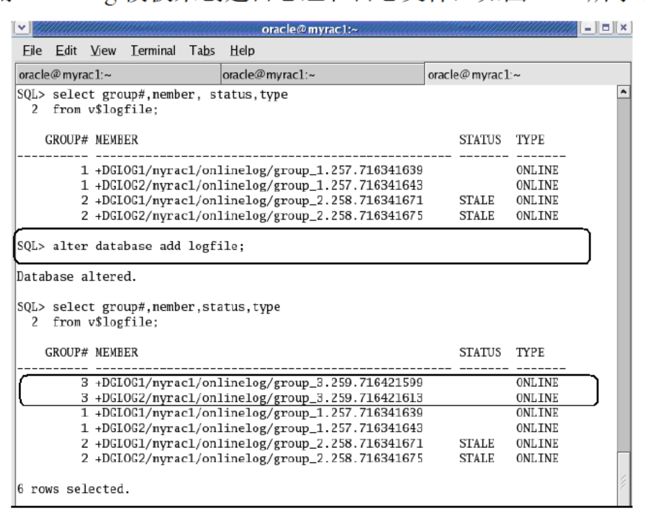


图 2-54　创建新的重做日志组

在上图中，我们使用数据字典视图 v$logfile 查询当前的日志组和相应的文件信息，可以看到有两个日志组，每个日志组两个日志成员。



我们使用 alter database add logfile 来创建新的日志组，以及为日志组添加成员，一行指令却做了“大量”工作，这就是 ASM 使用 ONLINELOG 模板的结果。

2.11　使用RMAN将数据库迁移到ASM实例

在上面的几节我们对 ASM 已经有了充分的理解，下面我们通过一个迁移过程，说明如何将非 ASM 存储管理的数据库迁移到一个 ASM 实例上，我们在单实例环境下完成这个迁移任务。

迁移之前的环境是单实例数据库，ORACLE_SID 为 myrac1，系统采用本地文件系统，所有文件采用安装数据库时的默认设置，其文件分布如图 2-55 所示。

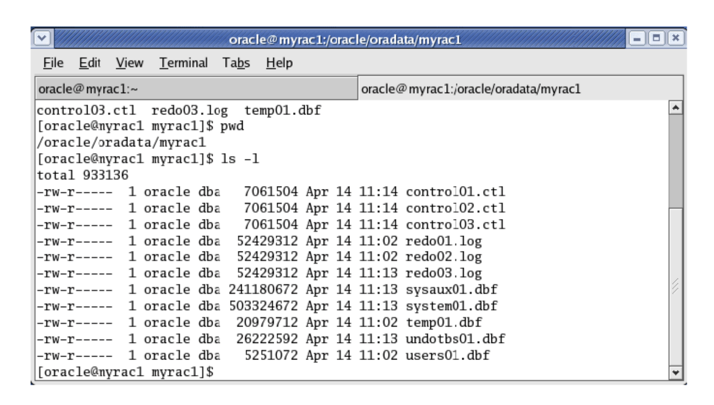


图 2-55　迁移前数据库文件分布

ASM 实例的环境，有四个磁盘组 DATA、RECOVERY、DGLOG1、DGLOG2，且磁盘组类型均为为 EXTERNAL，每个磁盘组一个 ASM 磁盘，磁盘对应的物理分区分别为/dev/sdb3、/dev/sdb4、/dev/sdc1、/dev/sdc2（创建四个磁盘组）。其实这也就是 2.5 节手工创建的 ASM 实例环境。

下面是迁移的具体步骤，我们使用 RMAN 工具完成迁移。

01 修改 SPFILE 参数，这里需要在 SPFILE 参数中增加三个参数以及相应的值。添加参数的内容如下例所示。

```
DB_CREATE_FILE_DEST='+DATA'
DB_RECOVERY_FILE_DEST='+RECOVERY'
DB_RECOVERY_FILE_DEST_SIZE=3g
DB_CREATE_ONLINE_LOG_DEST_1=DGLOG1
DB_CREATE_ONLINE_LOG_DEST_2=DGLOG2
```

实际上 ASM 使用的是 OMF 文件系统，所以必须设置这些参数。下面分别介绍这四个参数的含义：

- DB_CREATE_FILE_DEST：该参数指定了数据文件、控制文件、重做日志文件以及临时文件的默认创建位置，如果没有设置 DB_CREATE_ONLINE_LOG_DEST_n 参数，会在参数 DB_CREATE_FILE_DEST 指定的磁盘组中创建重做日志文件和控制文件。
- DB_RECOVERY_FILE_DEST：该参数指定了 RMAN 备份、闪回日志以及归档日志的存储位置，如果没有设置参数 DB_CREATE_ONLINE_LOG_DEST_n，则会在 DB_RECOVERY_FILE_DEST 参数指定的目录下创建一个重做日志文件。
- DB_RECOVERY_FILE_DEST_SIZE：该参数设置快闪恢复区的大小。
- DB_CREATE_ONLINE_LOG_DEST_n：该参数存储重做日志和控制文件的副本，实现多路复用。

接着我们演示如何修改这四个参数，首先需要修改 ORACLE_SID 登录实例名为 myrac1 的数据库系统，如图 2-56 所示。

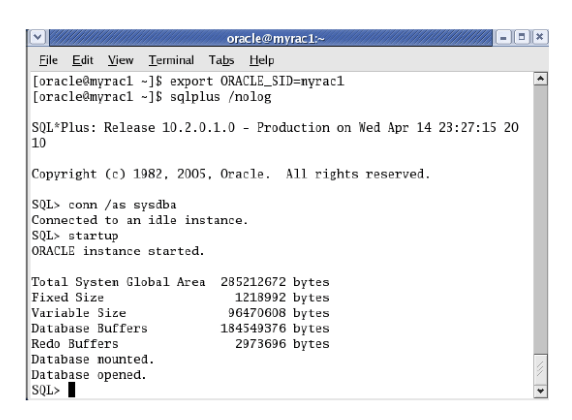


图 2-56　启动数据库

然后修改 spfile 参数文件，如图 2-57 所示。

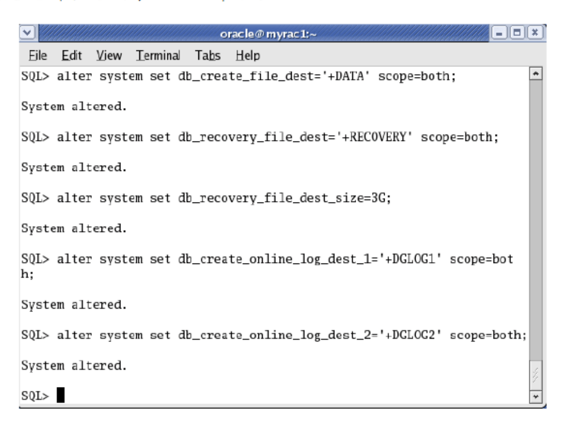


图 2-57　修改 spfile 参数

02 把控制文件迁移到 ASM。

首先要通过 spfile 创建 pfile，目的是修改 pfile 中控制文件的参数然后创建 spfile，这样就达到修改 spfile 参数文件的目的。先创建 pfile，如下例所示。

例子 2-24 从 SPFILE 创建 PFILE。

```
SQL> create  pfile='/oracle/initmyrac1.ora'  from spfile;
File created.
```

然后删除 pfile 内的控制文件参数。关闭数据库，再执行如下指令。

例子 2-25 关闭数据库并从 pfile 创建 spfile。

```
SQL>shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> create spfile from pfile='/oracle/initmyrac1.ora';
File created
```

接着启动数据库到 nomount 状态，使用 RMAN 恢复控制文件到新的 ASM 磁盘位置，如图 2-58 所示。

在图 2-58 中，控制文件在 ASM 磁盘组 DGLOG1 和 DGLOG2 中分别创建，实现了多路复用。

03 将数据库文件复制到 ASM 磁盘组。我们还是使用 RMAN 工具，此时需要把数据库启动到 MOUNT 状态。

首先将数据库切换到 MOUNT 状态，如下例所示。

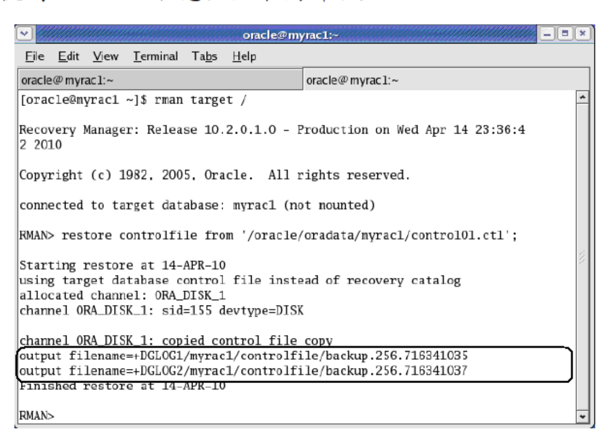


图 2-58　在 ASM 磁盘组中恢复控制文件

例子 2-26　将数据库切换到 MOUNT 状态。

```
RMAN>alter database mount;

Database mounted
Released channel: ORA_DISK_1
```

接着将数据库文件复制到 ASM 磁盘，如下例所示。

例子 2-27　将数据库文件复制到 ASM 磁盘。

```
RMAN>backup as copy database format '+DATA';

Starting backup at 14-APR-10
Allocated channel: ORA_DISK_1
Channel ORA_DISK_1:starting datafile copy
Input datafile fno=00001 name=/oracle/oradata/myrac1/system01.dbf
Output filename=+DATA/myrac1/datafile/system.257.716311037 tag=
    TAG20100414T151716
Recid=1 stamp=716311078
Channel ORA_DISK_1: datafile copy complete, elapsed time : 00:00:45
Channel ORA_DISK_1:starting datafile copy
Input datafile fno=00003 name=/oracle/oradata/myrac1/sysaux01.dbf
Output filename=+DATA/myrac1/datafile/sysaux.258.716311083 tag=
        TAG20100414T151716
Recid=2 stamp=716311105
Channel ORA DISK 1: datafile copy complete , elapsed time: 00:00:25
Channel ORA_DISK_1: starting datafile copy
```

```
Input datafile fno=00002 name=/oracle/oradata/myrac1/undotbs01.dbf
Output filename=+DATA/myrac1/datafile/undotbs1.259.716311109 tag=
        TAG20100414T151716
Recid=3 stamp=716311110
Channel ORA_DISK_1: datafile copy complete, elapsed time: 00:00:03
Channel ORA DISK 1: starting datafile copy
Input datafile fno=00004 name=/oracle/oradata/myrac1/users71601.dbf
Output filename=+DATA/myrac1/datafile/users.260.716311111 tag=
        TAG20100414T151716
Recid=4 stamp=716311112
Channel ORA DISK 1: datafile copy complete, elapsed time: 00:00:01
Channel ORA DISK 1:starting datafile copy
Copying current control file
Output filename=+DATA/myrac1/controlfile/backup.261.71631113 tag=
        TAG20100414T151716
Recid=5 stamp=716311114
Channel ORA DISK 1: datafile copy complete, elapsed time: 00:00:04
Channel ORA DISK 1: starting full datafile backupset
Channel ORA DISK 1: specifying datafile(s) in backupset
Including current SPFILE in backupset
Channel ORA DISK 1: starting piece 1 at 14-APR-10
Channel ORA DISK 1: finished piece 1 at 14-APR-10
Piece handle=+DATA/myrac1/backupset/
        2010 04 14/nnsnf0 tag20100414t151716 0.262.716311117
Tag= TAG20100414T151716 comment=NONE
Channel ORA DISK 1: backup set complete, elapsed time: 00:00:01
Finished backup at 14-APR-10

RMAN>
```

04 将所有的数据文件转换到 ASM 磁盘组+DATA 中，如下例所示。

例子 2-28 将所有的数据文件转换到 ASM 磁盘组+DATA。

```
RMAN> switch database to copy

datafile 1 switched to datafile copy "+DATA/myrac1/datafile/
    system.257.716311037
datafile 2 switched to datafile copy "+DATA/myrac1/undotbs1.259.716311109
datafile 3 switched to datafile copy "+DATA/myrac1/datafile/
    sysaux.258.716311083
datafile 4 switched to datafile copy "+DATA/myrac1/users.260.716311111
```

05 打开数据库，如下例所示。

例子 2-29 在 RMAN 环境下打开数据库。

```
RMAN> alter database open;

Database opened
RMAN>
```

06 将重做日志迁移到 ASM 磁盘组。

首先，我们先确认当前系统日志文件的位置，如图 2-59 所示。

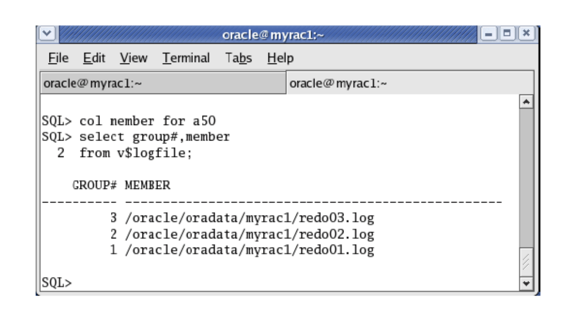


图 2-59　当前日志文件位置

然后进入 RMAN，通过“alter database add logfile”指令把日志文件迁移到 ASM 系统中，此时数据库是打开的，具体操作如下例所示。

例子 2-30　使用 RMAN 指令迁移日志文件到 ASM 系统。

```
RMAN> sql "alter database add logfile member '+DGLOG1', 'DALOG2'  to group 1";
sql statement: alter database add logfile member '+DGLOG1', 'DALOG2'  to group 1;
RMAN> sql "alter database add logfile member '+DGLOG1', 'DALOG2'  to group 2";
sql statement: alter database add logfile member '+DGLOG1', 'DALOG2'  to group 2;
```

此时会在日志组 1 和日志组 2 中各增加两个日志成员，每个日志组中新增的日志成员分别存储在磁盘组 DGLOG1 和 DGLOG2 中。我们查询当前的日志组信息，如图 2-60 所示。

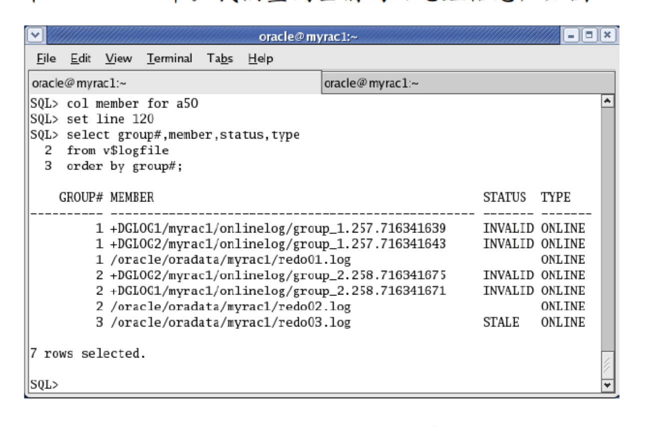


图 2-60　当前日志组信息

显然从上述查询结果中可以看出日志组 1 和日志组 2 都有三个日志成员，都有两个日志成员分布在 ASM 磁盘组中。

然后，删除原来的重做日志文件。在删除之前必须保证该日志文件不是当前正在使用的日志组，并且保证 ASM 磁盘组 DGLOG1 和 DGLOG2 中的日志成员类型都为 ONLINE。

在下图 2-61 中，我们首先使用 alter system switch logfile 指令切换日志组，这个过程可能需要执行几次。使得磁盘组中的日志成员为有效的成员。

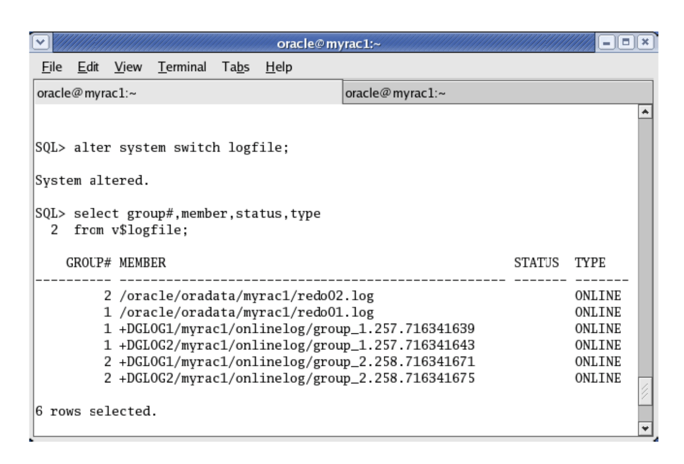


图 2-61　将磁盘组中的日志成员置为有效

接下来删除基于系统文件的日志成员，依次执行如下例所示的两条指令。

例子 2-31 删除基于文件系统的日志成员

```
SQL> alter database drop logfile member '/oracle/oradata/myrac1/redo01.log';
SQL> alter database drop logfile member '/oracle/oradata/myrac1/redo02.log';
```

如果遇到要删除的日志成员是当前正在使用的日志组，则使用 alter system switch logfile 指令来切换日志组后再执行删除日志成员的操作。



Oracle 要求至少有两个重做日志组，每个日志组至少有两个日志成员，所以我们在迁移日志文件到 ASM 时，必须保证满足这个条件。

在删除完成后，我们查看当前的日志组状态，如图 2-62 所示。

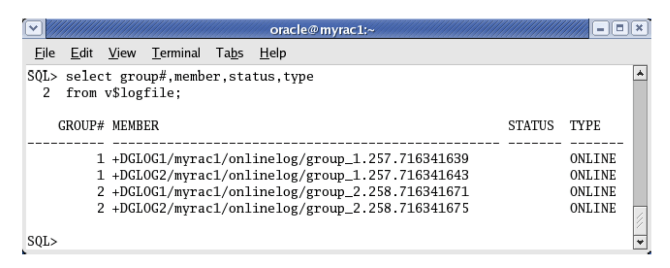


图 2-62　查看 ASM 磁盘组中的重做日志组状态

从上图可以看出，重做日志都已经成功迁移到了 ASM 磁盘组 DGLOG1 和 DGLOG2 中了。

07 在 ASM 磁盘组中创建临时文件。因为 RMAN 不会迁移临时文件，所以需要在 ASM 磁盘组中手动创建一个临时表空间。我们先查看当前临时文件的信息，如图 2-63 所示。

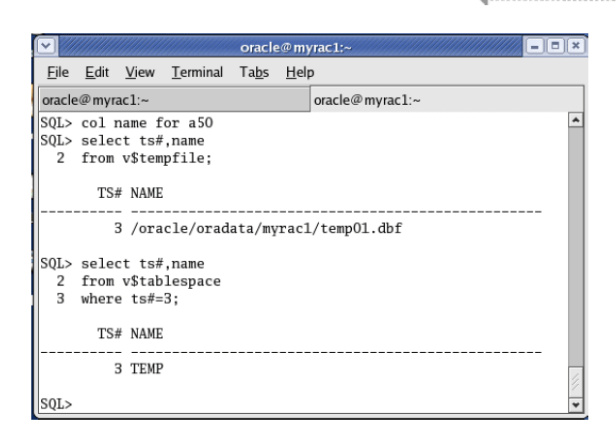


图 2-63　当前系统临时文件信息。

下面我们使用 RMAN 在 ASM 磁盘组 DATA 重建一个临时表空间。如下例所示。

例子 2-32　在 ASM 磁盘组 DATA 重建一个临时表空间。

```
RMAN>sql "alter tablespace temp add tempfile size 1024m";
Sql statement: alter tablespace temp add tempfile size 1024m;
RMAN>
```

然后，我们需要删除旧的临时文件，如下例所示。

例子 2-33　删除旧的临时文件。

```
RMAN>sql "alter database tempfile '/oracle/oradata/myrac1/temp01.dbf' drop ";
Sql statement: alter database tempfile '/oracle/oradata/myrac1/temp01.dbf'
drop;
```

接着我们验证删除结果，以及是否在 ASM 磁盘 DATA 成功创建了临时文件，如图 2-64 所示。

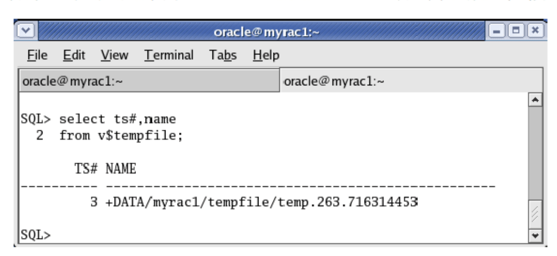


图 2-64　查看当前 ASM 磁盘上的临时文件

从上图的输出可以看出，已经在 ASM 磁盘 DATA 成功创建了一个临时文件，同时也删除了旧的临时文件。

2.12　管理ASM的数据字典视图

在管理 ASM 实例的过程中，往往需要通过视图更深入地了解 ASM 实例当前的状态，以及和数据库实例的关系。下面是我们在维护 ASM 实例过程中经常使用的数据字典视图，读者可以大致

了解其作用，自己去查询视图的结构，通过实际操作学习这些视图的用处。下面给出常用视图的一个概览表，读者可以参考使用，如表 2-2 所示。

表 2-2　管理 ASM 的数据字典视图

视图名	视图功能解释
V$ASM_DISKGROUP	该视图描述磁盘组组号，组名，磁盘大小，状态以及冗余类型等信息
V$ASM_DISK	该视图描述了 ASM 实例发现的每一个磁盘，其中也包括不属于任何磁盘组的磁盘
V$ASM_FILE	在 ASM 实例中发现的磁盘组中的 ASM 文件。前提是该磁盘组已经是 mount 状态
V$ASM_ALIAS	该视图记录磁盘组中的每个别名信息
V$ASM_TEMPLATE	该视图记录了 ASM 实例挂接的磁盘组中的所有模板信息，如模板名称，冗余度以及条带粒度等信息
V$ASM_CLIENT	该视图描述使用 ASM 实例管理数据库文件的数据库信息，如数据库实例名，当前与 ASM 实例的连接状态等

2.13　本章小结

本章我们主要讲解了 ASM 磁盘存储的优势，以及 Oracle 实现 ASM 自动存储优势的背后技术支持，即自动冗余和条带化技术。接着我们重点介绍了 ASM 系统架构和实例架构，这样读者就从宏观上和实例级上理解 ASM 的本质，如何启动和关闭 ASM 实例是 DBA 维护必不可少的内容，我们都给出了详细的例子。

为了管理 ASM 实例以及文件系统，我们需要了解 ASMCMD 中指令集的内容，并掌握几个关键的指令如 mkdir、mkalia、lsct 等。然后介绍了 ASM 磁盘组管理和 ASM 文件管理，这也是 ASM 自动存储管理非常重要的内容。最后我们通过一个单实例数据库迁移到 ASM 的实例，详细介绍了迁移过程，使得读者对本章的知识点有个全局的把握和深入的理解。

第 3 章

管理Clusterware组件及管理指令

本章介绍如何管理 Clusterware 组件。我们知道 Clusterware 是 RAC 的核心，它是 RAC 的"基础设施"，负责管理集群中的硬件资源，将多个节点虚拟成一个服务器对外提供应用服务，对内提供各种组件的监控和管理任务。而 Voting Disk 和 OCR 是 Clusterware 集群件的重要组件，我们将学习这两个组件作用，维护这两个核心组件的方法以及常用的维护 Clusterware 的各种指令操作。

3.1 Clusterware及其组件

Clusterware 不仅可以为 Oracle 数据库提供高可用性的集群件，而且也可以为其他应用提供高可用性。一个节点的应用故障时，它保证自动切换到另一个可用的节点上，对用户提供了应用系统的可用性。

在 OracleRAC 环境下，Clusterware 监视集群中所有的组件如 instance、Listener，ONS、VIP、GSD 等。当这些组件故障时，集群会自动重启失败的组件或者将应用重定向到可用的节点，使得集群管理的整个系统对外提供不间断的应用服务。

Oracle Clusterware 包括两个非常重要的集群组件：Voting Disk 和 OCR（Oracle Cluster Registry）。其中 Voting Disk 是一个用于管理节点成员关系的文件。OCR 也是一个文件，它用于管理集群以及 Oracle RAC 数据库配置信息。

在第 1 章已经知道，在 Clusterware 的安装过程中已经创建了 OCR 和 Voting Disk，并且将这两个文件存储在两个共享的裸设备上。注意，OCR 和 Voting Disk 必须存放在共享的设备上，这样集群中的多个节点都可以读取或更改集群配置。为了防止 Voting Disk 文件的单点失败问题，可以配置冗余磁盘。如果配置了冗余磁盘，则必须保证冗余磁盘的一半以上是可以访问的，假设冗余磁盘数为 5 个，则必须有 3 个磁盘可以被所有节点访问。而对于 OCR 文件，也可以配置冗余磁盘，这些冗余磁盘都由 Clusterware 自动维护，冗余磁盘中只要有一个磁盘可以访问即可，不像 Voting Disk 文件必须保证至少一半以上的冗余磁盘可以访问。

鉴于 Clusterware 中组件 Voting Disk 和 OCR 的重要性，下面我们介绍如何备份这两个文件，以及如果当前的文件损坏应该如何恢复。

3.2 备份和恢复Voting Disks

Voting Disk 文件不需要经常备份，因为一般的 RAC 环境在部署前已经做了详细的规划，已经预见到扩展性的需求，一般不会在系统运行过程中再增加节点或改变 Voting Disk 的配置。

一般在以下几种情况下需要备份 Voting Disk。

- Clusterware 安装之后。
- 集群中增加或删除一个节点后。
- 在执行了诸如 Voting Disk 的 add 或 delete 操作后。

在了解了何时备份 Voting Disk 之后，下面介绍如何备份 Voting Disk 的内容。我们采用两种方式备份，一种是采用 Voting Disk 裸设备名的方式，一种是使用 Voting Disk 块设备名方式。下面分别介绍。

1．使用 Voting Disk 裸设备名备份 Voting Disk

使用 Voting Disk 裸设备名备份 Voting Disk。此时必须使用 root 用户登录，如下例所示。

例子 3-1 备份 Voting Disk 磁盘。

```
[root@myrac1~]# dd if=/dev/raw/raw3 of =/tmp/vddiskbackup2.dmp
196576+0 records in
196576+0 records out
```

在上述指令中，if 后紧跟的是裸设备名，而 of 后紧跟的是备份后的文件名。

2．使用 Voting Disk 块设备名备份 Voting Disk

使用 Voting Disk 块设备名备份 Voting Disk，如下例所示。

例子 3-2 备份 Voting Disk 磁盘。

```
[root@myrac1~]# dd if=/dev/sdc1 of=/tmp/vtdiskbackup.dmp
196576+0 records in
196576+0 records out
```

在备份完 Voting Disk 后，我们查看目录/tmp 下是否存在该文件，如图 3-1 所示。我们看到这两个备份后的文件大小相同，说明不论是使用裸设备名还是使用块设备名备份结果是一样的。

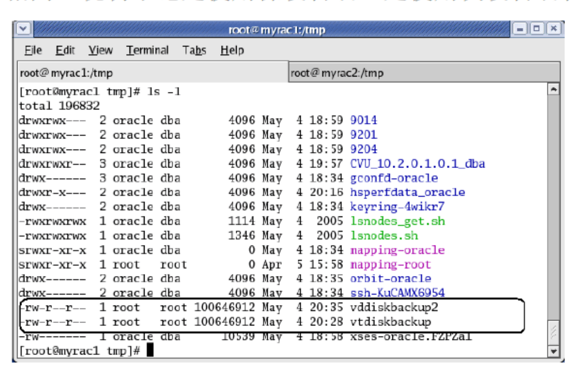


图 3-1 验证备份文件是否存在

从上图可以看出，我们成功备份了在例子 3-2 中的 Voting Disk 磁盘文件，如果 Voting Disk 的磁盘内容没有变化，在需要恢复的时候，就可以使用该备份文件执行恢复任务，此时不必停止 crsd.bin 进程，但必须保证 CRS 进程是启动的。恢复过程的操作步骤如下。

01 检查 CRS 状态，如下例所示。

例子 3-3 检查 CRS 各模块。

```
[root@myrac1 bin]# ./crsctl lsmodules crs
The following are the CRS modules::
CRSUI
CRSCOMM
CRSRTI
CRSMAIN
CRSRES
CRSCOMM
CRSOCR
CRSTIMER
CRSEVT
CRSD
CLUCLS
CSSCLNT
COMMCRS
COMMNS
```

02 确认备份文件位置后使用 dd 实现恢复，如下例所示。

例子 3-4 恢复 Voting Disk 文件。

```
[root@myrac1 bin]#dd if=/tmp/vtdiskbackup  of=/dev/sdc1
```



在使用 dd 指令备份和恢复 Voting Disk 文件时，if 表示输入文件，而 of 就表示输出文件。在备份时 if 表示要备份的文件位置，of 表示备份后的文件位置和文件名；而恢复时 if 表示备份后的文件位置，而 of 表示要恢复的文件位置。

3.3　添加和删除Voting Disks

在安装 RAC 时，有一项内容需要配置 Voting Disk 磁盘，在第一章中我们选择了使用 External Redundancy 的方式，如图 3-2 所示。

显然使用这种方式可靠性差，因为没有冗余备份，所以为了提高可靠性，我们可以添加 Voting Disk 磁盘。

在安装 RAC 配置 Voting Disk 磁盘时，我们也可以选用 Normal Configuration 选项，此时需要提供三个磁盘位置，用于存储 Voting Disk 文件，如图 3-3 所示。

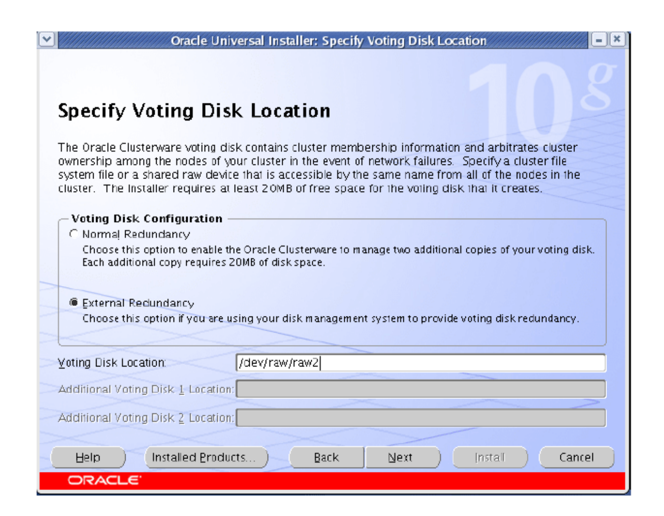


图 3-2 设置 Voting Disk 存储（External Redundancy）

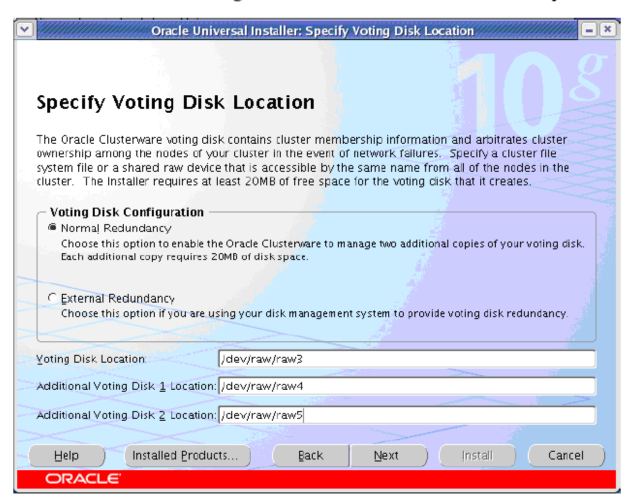


图 3-3 设置 Voting Disk 存储（Normal Redundancy）

而如果使用 Normal Redundancy 选项，则默认要求配置 Voting Disk 的另外两个文件拷贝位置，即提供该文件的冗余备份。

在我们选择使用 External Redundancy 后，如果出于高可用性的考虑可以使用增加 Voting Disk 磁盘的方法来实现冗余，当然在选择 Normal Redundancy 选项设置 Voting Disk 存储后也可以添加 Voting Disk 磁盘。

增加 Voting Disk 的前提条件是必须在两个节点上关闭 Oracle Clusterware，并且切换到 root 用户，下面演示如何添加 Voting Disk 磁盘。

01 在两个 RAC 的所有节点上执行如下指令关闭 Clusterware。

例子 3-5 关闭 Clusterware。

```
[root@myrac1 ]#cd /oracle/product/crs/bin
[root@myrac1 bin]# ./crsctl stop crs
```

```
Stopping resources.
Successfully stopped CRS resources
Stopping CSSD.
Shutting down CSS daemon.
Shutdown request successfully issued.
```

02 增加 Voting Disk 磁盘，如下例所示。

例子 3-6 增加 Voting Disk 磁盘。

```
[root@myrac1 ]#cd /oracle/product/crs/bin
[root@myrac1 bin]# ./crsctl add css /dev/raw/raw6 -force
Now formatting voting disk: /dev/raw/raw6
successful addition of votedisk /dev/raw/raw6.
```

和添加 Voting Disk 相对应，我们也可以删除不需要的 Voting Disk 磁盘。此时也需要首先关闭 Clusterware，并且使用-force 参数强制在 Clusterware 关闭时，修改 Voting Disk 的存储配置。如下例所示。

例子 3-7 删除 Voting Disk 磁盘。

```
[root@myrac1 ]#cd /oracle/product/crs/bin
[root@myrac1 bin]# ./crsctl delete css /dev/raw/raw6 -force
successful deletion of votedisk /dev/raw/raw6
```



如果需要在 Clusterware 关闭的时候，修改 Voting Disk 的内容，可以使用 force 选项。此时可以在不启动 Clusterware 的前提下，强制将修改的内容写入 Voting Disk 磁盘。

3.4 备份和恢复OCR

在配置 OCR 时也有两种选择，一种是 Normal Redundancy 方式，一种是 External Redundancy 方式，采用 Normal Redundancy 方式，如图 3-4 所示。

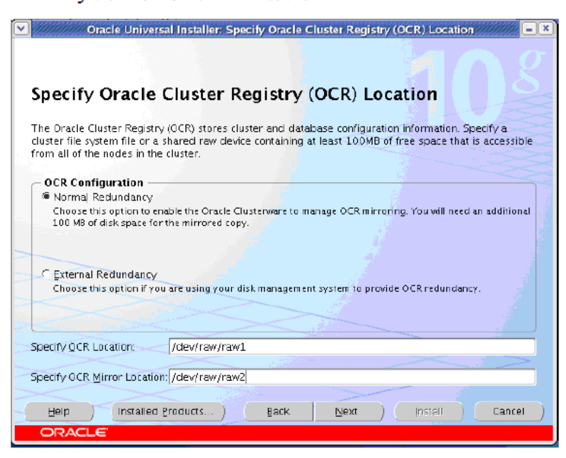


图 3-4　Normal Redundancy 方式配置 OCR

在上图中，需要制定两个 OCR 的存储位置，其中一个作为镜像位置。在选择了这种存储 OCR 的方式后，Oracle 的 Clusterware 会每 4 小时自动备份 OCR，如果存在多个备份，Clusterware 总会保存最近的三份备份。对于备份频率和备份的文件数量用户无法修改，都由 Clusterware 自己维护。自动备份的文件保存在$CRS_HOME/cdata/cluster_name 目录下，在笔者的电脑上 CRS_HOME 为/oracle/product/crs，而 cluster_name 为 crs，所以自动 OCR 备份的存储目录为/oracle/product/crs/cdata/crs。

对于这种备份方式，我们可以使用 ocrconfig –showbackup 指令查看当前的自动备份的文件内容，如下例所示。

例子 3-8 查看当前自动备份的文件。

```
[root@myrac1 ]#cd /oracle/product/crs/bin
[root@myrac1 bin]# ./ocrconfig -showbackup
```

另一种是没有镜像的配置，称为 External Redundancy 方式，如图 3-5 所示。

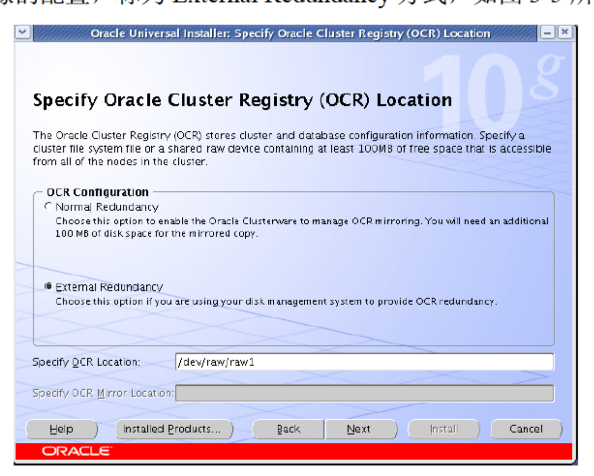


图 3-5 设置 OCR 存储

无论哪种方式，都可以采取主动的 OCR 文件备份方式，下面给出备份 OCR 文件的示例。

例子 3-9 备份 OCR 文件。

```
[root@myrac2 ~]# cd /oracle/product/crs/bin
[root@myrac2 bin]# ./crsconfig -export /tmp/ocrmybackup20100504
```

备份后，我们进入到/tmp 目录下查看是否创建成功，如图 3-6 所示。

有备份就有恢复，不然就失去了备份的意义。我们已经讨论过 OCR 可以有两种配置方式，一种是使用镜像（Normal Redundancy），一种不使用镜像（External Redundancy）。使用 Normal Redundancy 方式 Oracle 会自动备份，不需要人工干预（也无法干预☺），而后一种 External Redundancy 方式可以采用备份文件的方式人工备份。对应两种 OCR 的配置方式，我们有对应的 ORC 文件恢复方式，下面分别介绍。

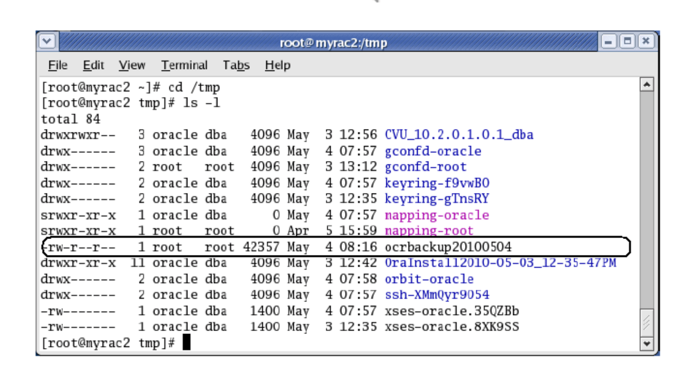


图 3-6　验证 OCR 文件备份

1．从自动备份中恢复 OCR

从自动备份中恢复 OCR 的详细步骤如下。

01 确认已经备份了 OCR 文件，如下例所示。

例子 3-10 验证 OCR 备份。

```
[root@myrac2 ~]# cd /oracle/product/crs/bin
[root@myrac2 bin]# ./ocrconfig -showbackup

…
```

02 关闭 Oracle Clusterware，如下例所示。

例子 3-11 关闭 Clusterware。

```
[root@myrac1 bin]# ./crsctl stop crs
Stopping resources.
Successfully stopped CRS resources
Stopping CSSD.
Shutting down CSS daemon.
Shutdown request successfully issued.
```



在两个节点上都需要关闭 Clusterware。

03 从第一步的备份文件中选择一个用于恢复 OCR 的备份文件，实现 OCR 的恢复，如下例所示。

例子 3-12 恢复 OCR。

```
[root@myrac1 bin]# ./ocrconfig -restore file_name
```

04 启动 Oracle Clusterware，如下例所示。

例子 3-13 启动 Clusterware。

```
[root@myrac1 bin]# ./crsctl start crs
Attempting to start CRS stack
The CRS stack will be started shortly
```

2. 从人工备份文件中恢复。

从人工备份文件中恢复同样需要如下几个步骤。

01 验证 OCR 的备份信息。我们在手工备份中已经使用 ocrconfig –export 指令备份了 OCR 文件，这里需要确认该文件没有损坏，而且存储备份的磁盘可以正常访问。

02 关闭 Oracle Clusterware，如下例所示。

例子 3-14 关闭 Clusterware。

```
[root@myrac1 bin]# ./crsctl stop crs
Stopping resources.
Successfully stopped CRS resources
Stopping CSSD.
Shutting down CSS daemon.
Shutdown request successfully issued.
```



在两个节点上都需要关闭 Clusterware。如果在没有关闭 Clusterware 的前提下，试图恢复 OCR 会出现如下错误。

```
[root@myrac2 bin]# ./crsconfig -import /tmp/ocrbackup20100405
PROT-19: Cannot proceed while clusterware is running . Shutdown clusterware
first
```

03 恢复 OCR 文件，如下例所示。

例子 3-15 恢复 OCR 文件。

```
[root@myrac1 bin]# ./ocrconfig -import /tmp/ocrbackup20100405
```

04 启动 Oracle Clusterware，如下例所示。

例子 3-16 启动 Clusterware。

```
[root@myrac1 bin]# ./crsctl start crs
Attempting to start CRS stack
The CRS stack will be started shortly
```

3.5 修改OCR存储配置信息

在安装 RAC 时，确定了 OCR 的存储位置后，可以修改 OCR 的存储配置，如增加 OCR 的存储、替换当前的 OCR 存储以及增加 OCR 映像存储等。下面分别介绍。

1. 增加 OCR 的存储位置

如果我们使用了 External Redundancy 方式来存储 OCR，在创建 RAC 后可以使用 ocrconfig 指令增加一个 OCR 的存储位置。如果使用 Normal Redundancy 方式，那没有必要再添加一个 OCR 的存储位置。但是 Oracle Clusterware 最多支持两个 OCR 存储位置。

下面例子演示如何添加一个 OCR 的存储位置，注意这个操作的前提是使用了 External Redundancy 方式来存储 OCR。

例子 3-17　添加一个 OCR 的存储位置

```
[root@myrac1 bin]# ./ocrconfig -replace ocr /dev/raw/raw6
```

此种情况下，也可以添加 OCR 镜像位置来实现 OCR 的冗余，如下例所示。

例子 3-18　添加 OCR 的镜像位置。

```
[root@myrac1 bin]# ./ocrconfig -replace ocrmirror /dev/raw/raw6
```

2．替换 OCR 的存储位置

如果在 OCR 的存储被损坏，或者打算改变当前 OCR 的存储位置，可以采用替换 OCR 存储位置的方式，具体指令为 ocrconfig –replace ocr disk。

当前的 OCR 存储在裸设备/dev/raw/raw1 和/dev/raw/raw2 上，下面演示如何替换当前的 OCR 存储位置。

01　首先运行 ocrcheck 查看当前的 OCR 存储信息，如图 3-7 所示。

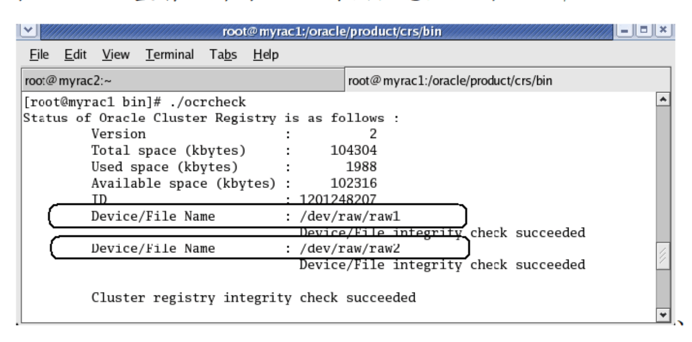


图 3-7　查看当前 OCR 的存储信息

从上图可以看出当前的 OCR 存储在两个裸设备上，一个为/dev/raw/raw1，一个为/dev/raw/raw2。/dev/raw/raw1 为 OCR 的存储位置，/dev/raw/raw2 为 OCR 的镜像位置。接下来我们会修改 OCR 的这个存储位置。

02　在使用 ocrconfig 指令前必须保证 Cluster 是运行的，下面通过指令 crsctl 来查看当前 Cluster 的运行状态。

例子 3-19　查看 Cluster 的运行状态。

```
[root@myrac1 bin]# ./crsctl check crs
CSS appears healthy
CRS appears healthy
EVM appears healthy
```

03　修改 OCR 的这个存储位置，如下例所示。

例子 3-20　修改 OCR 的存储位置。

```
[root@myrac1 bin]# ./ocrconfig -replace ocr /dev/raw/raw6
```

此时，我们将 OCR 的存储位置从/dev/raw/raw1 修改为/dev/raw/raw6。

04　验证修改结果。

下面通过 ocrcheck 指令查看修改结果，如图 3-8 所示。

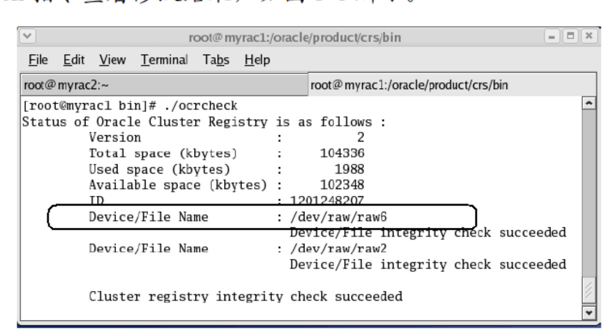


图 3-8　查看 OCR 的存储位置修改结果

上图中可以看出当前 OCR 的存储位置已经修改为裸设备/dev/raw/raw6。

3．替换 OCR 镜像的存储位置

替换 OCR 镜像的存储位置的步骤和替换 OCR 的存储位置的步骤相同，这里不再重复，只要读者有一个裸设备，即可通过如下例所示的方法修改。

例子 3-21　修改 OCR 的镜像存储位置。

```
[root@myrac1 bin]# ./ocrconfig -replace ocrmirror /dev/raw/raw6
```

3.6　删除OCR存储

有时我们将 OCR 存储在了像 RAID 这样的冗余存储上，这样我们就没有必要保留多个 OCR 映像，此时就可以删除一个 OCR 存储。

首先使用 ocrcheck 查看当前的 OCR 存储信息，如图 3-8 所示。接着删除一个 OCR 的备份存储，如下例所示。

例子 3-22　删除 OCR 的备份存储。

```
[root@myrac1 bin]# ./ ocrconfig -replace ocr
```

再次使用 ocrcheck 查看当前的 OCR 存储信息，如图 3-9 所示。

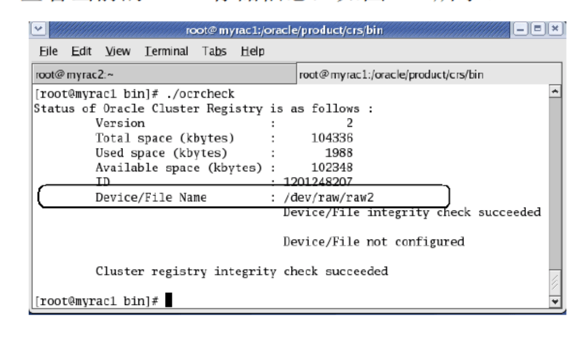


图 3-9　验证 OCR 备份存储的删除结果

在上图中的 OCR 只有一个存储位置即/dev/raw/raw2，说明已经使用 ocrconfig –replace ocr 指令删除了 OCR 的备份存储。

3.7　ocrconfig指令功能汇总

我们可以在$OCR_HOME/bin 目录下输入./ocrconfig，然后回车来查看该指令的和使用方法，如下例所示。

例子 3-23 指令 ocrconfig 用法。

```
[root@myrac2 bin]# ./ocrconfig
Name:
        Ocrconfig -a Configuration tool for Oracle Cluster Registry
Synopsis:
      Ocrconfig [option]
      Option:
         -export <filename> [-s online]
                                -Export cluster register contents to a file
         -import <filename>     -Import cluster registry contents from a file
         -update [<user>[<group>]]
                            -Update cluster registry from previous version
            -downgrade[-version <version string>]
                            -Downgrade cluster registry to the specified version
            -backuploc<dirname>      -Configure periodic backup location
            -showbackup              -Show backup information
            -restore <filename>      -Restore from physical backup
            -replac ocr|ocrmirror [<filename>]
                                     -Add/replace/remove a OCR device/file
            -overwrite               -Overwrite OCR configuration on disk
            -repair ocr|ocrmirror <filename>
                                     -Repair local OCR configuration
            -help                    -Print out this help information
Note:
      A log file will be created in
      $ORACLE_HOME/log/<hostname>/client/ocrconfig_<pid>.log. Please ensure
you have file creation privileges in the above directory before running this tool.
```

上面的多数参数，我们在 3.5 节、3.6 节和 3.7 节已经使用过，这里再介绍一下-repair 参数的含义。

在我们修改 OCR 的存储配置时，如添加、替换或者删除一个 OCR 等操作，如果此时该节点宕机，就需要在该节点上使用-repair 参数来修复 OCR 的配置，其语法如下例所示。

```
ocrconfig -repair ocrmirror device_name
```

如果 OCR 镜像的存储设备名为/dev/raw/raw6，则使用如下指令修改 OCR 的配置信息。

```
[root@myrac1 bin]# ./ ocrconfig -repair ocrmirror /dev/raw/raw6
```

3.8 管理Clusterware指令

管理 Clusterware 的命令位于$CRS_HOME/bin 目录下，我们可以通过 Linux 系统的 ls 命令查看可以用于维护 Clusterware 的指令，如图 3-10 所示。

```
oracle@myrac1:/oracle/product/crs/bin
File  Edit  View  Terminal  Tabs  Help
root@myrac2:~    oracle@myrac1:/oracle/product/crs/bin
[oracle@myrac1 bin]$ ls
cemutlo            crs_setperm             evmpost.bin     kgpmon          oifcfg
cemutlo.bin        crs_setperm.bin         evmshow         lbuilder        oifcfg.bin
cemutls            crs_start               evmshow.bin     lcsscan         olsadmintool
cemutls.bin        crs_start.bin           evmsort         linkshlib       olsnodes
clscfg             crs_stat                evmsort.bin     lmsgen          olsnodes.bin
clscfg.bin         crs_stat.bin            evmwatch        loadpsp         olsoidsync
clsfmt             crs_stop                evmwatch.bin    loadpsp0        onsctl
clsfmt.bin         crs_stop.bin            evt.sh          localconfig     oraenv
clsid              crstmpl.scr             extjob          lsdb            osdbagrp
clsid.bin          crs_unregister          extjobo         lsnodes         racgeut
cluutil            crs_unregister.bin      extjob0         lsnodes.bin     racgevtf
cluvfy             dbhome                  extjobo0        lxchknlb        racgmain
coraenv            diagcollection.pl       genagtsh        lxegen          racgons
crsctl             echodo                  genclntsh       lxinst          racgons.bin
crsctl.bin         evmd                    genclntst       oclsmon         racgvip
crsd               evmd.bin                genezi          oclsmon.bin     racgwrap
crsd.bin           evminfo                 gennfgt         ocrcheck        racgwrap.sbs
crs_getperm        evminfo.bin             gennttab        ocrcheck.bin    rawutl
crs_getperm.bin    evmlogger               genorasdksh     ocrconfig       sclsspawn
crs_profile        evmlogger.bin           gensyslib       ocrconfig.bin   srvctl
crs_profile.bin    evmmkbin                gsd             ocrdump         unzip
crs_register       evmmkbin.bin            gsdctl          ocrdump.bin     usrvip
crs_register.bin   evmmklib                gsd.sh          ocssd           vipca
crs_relocate       evmmklib.bin            kgmgr           ocssd.bin       xmlwf
crs_relocate.bin   evmpost                 kgmgr0          oerr            zip
[oracle@myrac1 bin]$
```

图 3-10 维护 Clusterware 的命令集合

我们要学习的管理 Clusterware 的指令都位于$CRS_HOME/bin 目录下。当我们执行这些指令完成维护任务时，需要告诉系统执行指令的具体位置。在本书中，我们都采用先切换到指令所在目录，再执行当前目录下的指令的方法，如下例所示。

```
[oracle@myrac1 ~]# cd $CRS_HOME/bin
[oracle@myrac1 bin]# ./crsctl check crs
```

这里先进入指令 crsctl check crs 所在目录，再执行该指令。

3.8.1 srvctl 指令

该指令的功能强大，可以控制 RAC 环境下的几乎所有对象，如数据库、实例、服务、节点、asm 以及监听器。

```
[root@myrac1 bin]# ./srvctl
Usage: srvctl <command> <object> [<options>]
   command : enable | disable | start | stop | relocate | status | add | remove
| modify | getenv | setenv | unsetenv | config
   objects: database | instance | service | nodeapps | asm | listener
For detailed help on each command and object and its options use:
Srvctl <command> <object> -h
```

指令 srvctl 的用法是 srvctl <command> <object> [<options>]，我们可以通过<command> 与<object>的组合来使用该指令管理 RAC 环境下的各种资源。下面我们给出几个常用的例子，这些

例子在实际的维护也很重要。

在使用这个指令时，需要一个技巧就是如何获得帮助信息，其实上述代码已经说明，只是没有具体的演示，比如我们要使用 srvctl config database 指令，此时如果想知道该参数后还有什么参数可以更细粒度地控制数据库的行为，可以紧跟-h 参数。下面我们根据对象 objects 分为 5 个部分来讲述 srvctl 的使用。

1．数据库（database）

使用 srvctl 来操作和数据库相关的行为。首先，查看当前数据库配置信息，我们先看如何获得帮助。

```
[oracle@myrac1 bin]$ ./srvctl config database -h
Usage :srvctl config database
Usage :srvctl config databae -d <name> [-a] [-t]
  -d <name>           Unique name for the database
  -a                  Additional attributes
  -t                  Display TNS entries
  -h                  Print usage
```

可见参数-d 后紧跟数据库标识，-a 参数将说明更多的信息，我们通过例子查看当前 RAC 环境下的数据库配置信息，如下例所示。

例子 3-24 查看数据库配置。

```
[oracle@myrac1 bin]$ ./srvctl config database -d myrac
myrac1  myrac1  /oracle/product/database
myrac2  myrac2  /oracle/product/database
```

从输出可以看出该 RAC 由两个节点 myrac1 和 myrac2 组成，在各自节点上的实例名分别为 myrac1 和 myrac2，随后的目录说明参数$ORACLE_HOME 的值。

```
[oracle@myrac1 bin]$ ./srvctl config database -d myrac -a
myrac1  myrac1  /oracle/product/database
myrac2  myrac2  /oracle/product/database
DB_NAME=myrac
ORACLE_HOME: /oracle/product/database
SPFILE: +DATA/myrac/spfilemyrac.ora
DOMAIN: null
DB ROLE: null
START_OPTIONS: null
POLICY:  AUTOMATIC
ENABLED FLAG: DB ENABLED
```

从上例的输出我们可以更清楚地了解数据库的信息，如$ORACLE_HOME、SPFILE 文件位置、数据库标识以及节点信息。

使用 srvctl 关闭和启动数据库。我们先看关闭数据库的更详细的参数设置。如下例所示。

```
[oracle@myrac1 bin]$ ./srvctl stop database -h
Usage : srvctl stop database -d <name> [-o <stop_option>] [-c <connect_str>
| q]
     -d <name>          Unique name for the database
     -o<stop_options>   Options to shtudown command(e.g. normal, transactional,
```

```
                      immediate,or abort)
    -c<connstr>       Connect string (default: / as sysdba)
    -q                Query connect string from standard input
    -h                Print usage
```

-d 参数紧跟数据库标识，而-o 参数说明关闭数据库的选项，有四个选项即 normal、transactional、immediate 和 abort，-c 参数指明连接字符串，-q 参数要求手动输入连接字符串，默认是/as sysdba。我们还是通过例子演示。

例子 3-25 使用 srvctl 关闭数据库。

```
[oracle@myrac1 bin]$ ./srvctl stop database -d myrac -o immediate
```

使用 srvctl 启动数据库。我们先看启动数据库的更详细的参数设置。如下例所示。

```
   [oracle@myrac1 bin]$ ./srvctl start database -h
   Usage : srvctl stop database -d <name> [-o <stop_option>] [-c <connect_str>
| q]
      -d <name>             Unique name for the database
      -o<start_options>     Options to startup command(e.g. open, mount, or mount)
      -c<connstr>           Connect string (default: / as sysdba)
      -q                    Query connect string from standard input
      -h                    Print usage
```

-d 参数紧跟数据库标识，而-o 参数说明启动数据库的选项，有三个选项即 open、mount 和 nomount，-c 参数指明连接字符串，-q 参数要求手动输入连接字符串，默认是/as sysdba。我们通过下面例子演示如何启动数据库。

例子 3-26 使用 srvctl 启动数据库。

```
[oracle@myrac1 bin]$ ./srvctl start database -d myrac -o open
[oracle@myrac1 bin]$
```

此时不会有任何提示信息，如果顺利启动则自动退出执行指令。

配置数据库自动启动特性，即在系统启动时自动启动数据库，可以使用 srvctl 的 enable 指令。我们先查看当前 RAC 环境下数据库是否是自动启动，如下例所示。

例子 3-27 查看当前数据库的配置信息。

```
[oracle@myrac1 bin]$ ./srvctl config database -d myrac -a
myrac1  myrac1  /oracle/product/database
myrac2  myrac2  /oracle/product/database
DB_NAME=myrac
ORACLE_HOME: /oracle/product/database
SPFILE: +DATA/myrac/spfilemyrac.ora
DOMAIN: null
DB_ROLE: null
START_OPTIONS: null
POLICY:  AUTOMATIC
ENABLED FLAG: DB DISABLED, INST DISABLED ON myrac1 myrac2
```

上例中输出的粗体字说明禁止了数据库的自动启动，下面通过例子启动数据库的自动启动。

例子 3-28　配置数据库自动启动。

```
[oracle@myrac1 bin]$ ./srvctl enable database -d myrac
```

接着我们查看数据库的配置信息，看是否成功启动了数据库的自动启动功能。如下例所示。

例子 3-29　查看是否成功配置数据库自动启动。

```
[oracle@myrac1 bin]$ ./srvctl config database -d myrac -a
myrac1  myrac1  /oracle/product/database
myrac2  myrac2  /oracle/product/database
DB_NAME=myrac
ORACLE_HOME: /oracle/product/database
SPFILE: +DATA/myrac/spfilemyrac.ora
DOMAIN: null
DB_ROLE: null
START_OPTIONS: null
POLICY:  AUTOMATIC
ENABLED FLAG: DB ENABLED
```

这样，当 CRS 启动后数据库将自动启动。如果不希望这样的行为，可以使用 disable 参数关闭数据库自动启动，如下例所示。

例子 3-30　关闭数据库自动启动。

```
[oracle@myrac1 bin]$ ./srvctl disable database -d myrac
```

删除数据库操作，使用 srvctl 指令可以轻易地删除数据库，如下例所示。

例子 3-31　删除数据库操作。

```
[oracle@myrac1 bin]$ ./srvctl remove database -d myrac
Remove the database myrac? (y/[n])
```

在手工恢复 OCR 时，需要向 OCR 中添加数据库对象，此时就需要使用 srvctl 的 add database 指令，如下例所示。

例子 3-32　向 OCR 添加 DATABASE 对象。

```
[oracle@myrac1 bin]$ ./ srvctl add database -d myrac -o /oracle/product/database
```

其中-d 参数说明添加的数据库对象，而-o 参数说明$ORACLE_HOME 变量的值。

2．ASM

首先查看 ASM 信息，如下例所示。

例子 3-33　查看 ASM 信息。

```
[oracle@myrac1 bin]$ ./srvctl config asm -n myrac2
+ASM2  /oracle/product/database
[oracle@myrac1 bin]$ ./srvctl config asm -n myrac1
+ASM1  /oracle/product/database
```

输出信息中+ASM2 为 ASM 实例名，而目录/oracle/product/database 为$ORACLE_HOME 目录。

例子 3-34 查看 ASM 实例状态信息。

```
[oracle@myrac1 bin]$ ./srvctl status asm -n myrac1
ASM instance +ASM1 is running on node myrac1.
[oracle@myrac1 bin]$ ./srvctl status asm -n myrac2
ASM instance +ASM2 is not running on node myrac1.
```

从输出看出实例 ASM2 没有运行，所以我们继续使用 srvctl 的 start 参数启动实例 ASM2，如下例所示。

例子 3-35 启动实例 ASM2

```
[oracle@myrac1 bin]$ ./srvctl start asm -n myrac2
```

上例我们也可以使用-i 参数指定节点 myrac2 上的实例名，但是由于该节点上只有一个实例，所以可以不使用-i 参数。

紧接着我们查询该实例是否运行，如下例所示。

例子 3-36 查实例 ASM2 是否运行。

```
[oracle@myrac1 bin]$ ./srvctl status asm -n myrac2
ASM instance +ASM2 is  running on node myrac1.
```

输出显示，我们成功启动了节点 myrac2 上的实例+ASM2。

下面我们使用 srvctl 的 enable 指令配置 ASM 实例自动启动。

例子 3-37 配置 ASM 实例自动启动。

```
[oracle@myrac1 bin]$ ./srvctl enable asm -n myrac1 -i +ASM1
```

关闭 ASM 实例，使用 srvctl 的 start 和 stop 指令，这里只给出一个操作指令。

例子 3-38 使用 srvctl 指令关闭 ASM 实例。

```
[oracle@myrac1 bin]$ ./srvctl stop asm -n myrac1 -i +ASM1
```

以上关闭了节点 myrac1 上的 +ASM1 实例。关闭 ASM 实例时，会自动关闭数据库，所以该指令还有-o 参数，选择关闭的方式，有四个选项即 normal、transactional、immediate 和 abort。

在手工恢复 OCR 时，需要恢复 ASM 实例对象，此时需要使用 srvctl 的 add instance 指令，如下例所示。

例子 3-39 向 OCR 中添加 ASM 对象。

```
[oracle@myrac1 bin]$ ./srvctl add asm -n myrac1 -i +ASM1
    -o /oracle/product/database
[oracle@myrac1 bin]$ ./srvctl add asm -n myrac2 -i +ASM2
    -o /oracle/product/database。
```

其中 –n 参数说明节点名，-i 说明 ASM 实例名，–o 参数说明$ORACLE_HOME。

至于 ASM 的其他操作读者可以自己尝试，如删除和添加 ASM，只要通过 –h 参数就能清楚地知道具体的操作，这里不再赘述。

3．监听器

首先查看各节点的监听器状态，如下例所示。

例子 3-40 查看各节点的监听器状态。

```
[oracle@myrac1 bin]$ ./srvctl config listener -n myrac1
Myrac1 LISTENER MYRAC1
[oracle@myrac1 bin]$ ./srvctl config listener -n myrac2
Myrac2 LISTENER_MYRAC2
```

输出显示节点名称，和各节点上对应的监听器名称。如节点 myrac1 上对应的监听器为 LISTENER_MYRAC1。

我们也可以使用 srvctl 指令来关闭与启动监听，如下例所示。

例子 3-41 关闭监听。

```
[oracle@myrac1 bin]$ ./srvctl stop listener -n myrac1
```

例子 3-42 启动监听。

```
[oracle@myrac1 bin]$ ./srvctl start listener -n myrac1
```

4．节点应用（nodeapps）

我们先查看节点应用的配置信息，可以使用 config nodeapps 指令。该指令相关的参数如下例所示。

```
 [oracle@myrac1 bin]$ ./srvctl config nodeapps -h
Usage : srvctl  config nodeapps -n <node_name> [ -a ] [ -g ][ -o ][ -s ][ -l ]
    -n <node>   Node name
    -a          Display VIP configuration
    -g          Display GSD configuration
  -s            Display ONS daemon configuration
  -l            Display listener configuration
  -h            Print usage
```

参数-a 显示 VIP 配置，参数-g 显示 GSD 配置，参数-s 显示 ONS 配置，参数-l 显示 listener 配置。下面通过一个例子演示参数-a、-g、-s、-l 参数的作用，如下例所示。

例子 3-43 显示节点应用资源信息。

```
[oracle@myrac1 bin]$ ./srvctl config nodeapps -n myrac1 - a
VIP exists.: /myrac1-vip/192.168.123.110/255.255.255.0/eth0
oracle@myrac1 bin]$ ./srvctl config nodeapps -n myrac1 - g
GSD exits.
[oracle@myrac1 bin]$ ./srvctl config nodeapps -n myrac1 - s
ONS daemon exists.
[oracle@myrac1 bin]$ ./srvctl config nodeapps -n myrac1 - l
Listener exists.
```

3.8.2　crs_stat 指令

该指令的作用是查看当前的 CRS 资源信息，以及控制这些资源的状态，如关闭和启动资源。

其使用方法如下例所示。

例子 3-44 查看指令 crs_stat 的使用方法。

```
[root@oracle  bin~]# ./crs_stat -help
Usage: crs_stat [resource_name [...]] [-v] [-l] [-q] [-c cluster_member]
     crs_stat [resource_name [...]] -t [-v] [-q] [-c cluster_member]
     crs_stat -p [resource_name [...]] [-q]
     crs_stat [-a] application -g
     crs_stat [-a] application -r [-c cluster_member]
     crs_stat -f [resource_name [...]] [-q] [-c cluster_member]
     crs_stat -ls [resource_name [...]] [-q]
```

下面分别通过示例说明如何使用这些指令以及指令输出的含义。

1. 使用-c 参数

查看节点 myrac2 上的资源，如下例所示。

例子 3-45 查看节点 myrac2 上的资源。

```
[root@oracle  bin~]# ./crs stat -c myrac2
NAME=ora.myrac.myrac2.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2

NAME=ora.myrac2.ASM2.asm
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2

NAME=ora.myrac2.LISTENER MYRAC2.lsnr
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2

NAME=ora.myrac2.gsd
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2

NAME=ora.myrac2.ons
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2
NAME=ora.myrac2.vip
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2
```

下面我们再查看节点 myrac1 的资源情况。

例子 3-46 查看节点 myrac1 上的资源。

```
[root@oracle bin~]# ./crs stat -c myrac1
NAME=ora.myrac.db
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1

NAME=ora.myrac.myrac1.inst
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1

NAME=ora.myrac1.ASM1.asm
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1

NAME=ora.myrac1.LISTENER_MYRAC1.lsnr
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1

NAME=ora.myrac1.gsd
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1

NAME=ora.myrac1.ons
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1
NAME=ora.myrac1.vip
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1
```

通过查看两个节点上的 CRS 资源，我们知道无论是哪个节点都有如下的资源，即 ASM 实例、数据库实例、监听器、GSD、ONS 以及 VIP。不同的是在节点 myrac1 上多了一个数据库资源，即 ora.myrac.db。

2．使用-f 参数

如果需要查询某个资源的详情，可以使用-f 参数，如下例所示。

例子 3-47 查询资源 ora.myrac1.vip 的详细信息。

```
[oracle@myrac2 bin]$ ./crs_stat – f ora.myrac1.vip
NAME=ora.myrac1.vip
TYPE=application
ACTION_SCRIPT=/oracle/product/crs/bin/racgwrap
ACTIVE_PLACEMENT=1
AUTO_START=1
```

```
CHECK_INTERVAL=0
RESTART_ATTEMPTS=0
RESTART_COUNT=0
DESCRIPTION=CRS application for VIP on a node
FAILOVER_DELAY=0
FAILOVER_INTERVAL=0
HOSTING_MEMBERS=myrac1
OPTIONAL_RESOURCES=
PLACEMENT=favored
REQUIRED_RESOURCES=
SCRIPT_TIMEOUT=60
START_TIMEOUT=0
STOP_TIMEOUT=0
UPTIME_THRESHOLD=7d
USR_ORA_ALTER_NAME=
USR_ORA_CHECK_TIMEOUT=0
USR_ORA_CONNECT_STR=/ as sysdba
USR_ORA_DEBUG=0
USR_ORA_FLAGS=
USR_ORA_IF=eth0
USR_ORA_INST_NOT_SHUTDOWN=
USR_ORA_LANG=
USR_ORA_NETWORK=255.255.255.0
USR_ORA_OPEN_MODE=
USR_ORA_OPI=false
USR_ORA_PFILE=
USR_ORA_PRECONNECT=none
USR_ORA_SRV=
USR_ORA_START_TIMEOUT=0
USR_ORA_STOP_MODE=immediate
USR_ORA_STOP_TIMEOUT=0
USR_ORA_VIP=192.168.123.110
FAILURE_THRESHOLD=0
FAILURE_COUNT=0
TARGET=ONLINE
STATE=ONLINE on myrac1
```

从输出可见关于该资源的信息十分详细，但并不是该资源信息中每一个显示的参数都对该资源有效。

3. 使用-v 参数

如果认为上述结果过于繁琐，可以使用-v 参数查看该资源的信息，选择哪个参数完全取决于读者的需要。下面演示 crs_stat –v 指令的使用。

例子 3-48 使用-v 参数查看资源 ora.myac.db 的信息。

```
[oracle@myrac2 bin]$ ./crs_stat -v ora.myrac.db
NAME=ora.myrac.db
TYPE=application
RESTART_ATTEMPTS=1
RESTART_COUNT=0
FAILURE_THRESHOLD=1
```

```
FAILURE_COUNT=0
TARGET=ONLINE
STATE=ONLINE on myrac2
```

从上例可以看出通过-v 参数可以查看某资源的状态和历史信息，如资源类型、是否 ONLINE 状态、尝试重启次数以及重启计数等。这里要求首先知道资源名称，那么如何知道 OCR 中注册的资源名称呢。可以直接使用 crs_stat 指令，而不用输入任何参数，如下例所示。

例子 3-49　显示 OCR 中注册的所有资源。

```
[oracle@myrac2 bin]$ ./crs_stat
NAME=ora.myrac.db
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac1
……
……
……
NAME=ora.myrac2.vip
TYPE=application
TARGET=ONLINE
STATE=ONLINE on myrac2
```

输出结果是 crs_stat –c myrac1 以及 crs_stat –c myrac2 指令的输出总和。这里不再全部列出，读者可以参考指令 crs_stat –c myrac1 以及 crs_stat –c myrac2 的输出结果。

4．使用-ls 参数。

该参数的含义是查看资源权限定义，既可以查询所有资源的权限定义，也可以查询单个资源的权限定义。我们查看单个资源的 ora.myrac.db 的权限定义，如下例所示。

例子 3-50　查看资源 ora.myrac.db 的权限定义。

```
[oracle@myrac2 bin]$ ./crs_stat  -ls  ora.myrac.db
Name        Owner      Primary PrivGrp     Permission
---------------------------------------------------------------------------
ora.myrac.db  oracle      dba                rwxrwxr--
```

从输出可以看出资源 ora.myrac.db 的权限为 rwxrwxr--，即对于资源拥有者具有读写执行的权利，对于与拥有者相同用户组的用户也具有同等权利。

如果想知道所有资源的权限信息，就不必指定资源名称，此时默认查询所有注册的资源信息，如下例所示。

例子 3-51　查看所有资源的权限定义。

```
[oracle@myrac2 bin]$ ./crs_stat  -ls  ora.myrac.db
Name                Owner       Primary PrivGrp         Permission
---------------------------------------------------------------------------
ora.myrac.db        oracle         dba                  rwxrwxr--
ora…….c1.inst       oracle         dba                  rwxrwxr—
ora……..c2.inst      oracle         dba                  rwxrwxr—
ora….SM1.asm        oracle         dba                  rwxrwxr—
ora…….c1.lsnr      oracle         dba                  rwxrwxr—
```

```
ora.myrac1.gsd      oracle       dba                    rwxrwxr—
ora.myrac1.ons      oracle       dba                    rwxrwxr—
ora.myrac1.vip      root         dba                    rwxrwxr—
ora…SM2.asm         oracle       dba                    rwxrwxr—
ora……..C2.lsnr      oracle       dba                    rwxrwxr—
ora.myrac2.gsd      oracle       dba                    rwxrwxr—
ora.myrac2.ons      oracle       dba                    rwxrwxr—
ora.myrac2.vip      root         dba                    rwxrwxr—
```

5. 使用 –v –t 参数

可以使用-v -t 参数查看当前 OCR 中注册资源的运行状态，如下例所示。

例子 3-52 使用-v –t 参数查看当前 OCR 中注册资源的运行状态。

```
[root@myrac1 bin]# ./crs_stat -t -v
Name           Type          R/RA    F/FT    Target    State     Host

ora.myrac.db   application   0/1     0/1     ONLINE    ONLINE    myrac2
ora....c1.inst application   0/5     0/0     ONLINE    ONLINE    myrac1
ora....c2.inst application   0/5     0/0     ONLINE    ONLINE    myrac2
ora....SM1.asm application   0/5     0/0     ONLINE    ONLINE    myrac1
ora....C1.lsnr application   0/5     0/0     ONLINE    ONLINE    myrac1
ora.myrac1.gsd application   0/5     0/0     ONLINE    ONLINE    myrac1
ora.myrac1.ons application   0/3     0/0     ONLINE    ONLINE    myrac1
ora.myrac1.vip application   0/0     0/0     ONLINE    ONLINE    myrac1
ora....SM2.asm application   0/5     0/0     ONLINE    ONLINE    myrac2
ora....C2.lsnr application   0/5     0/0     ONLINE    ONLINE    myrac2
ora.myrac2.gsd application   0/5     0/0     ONLINE    ONLINE    myrac2
ora.myrac2.ons application   0/3     0/0     ONLINE    ONLINE    myrac2
ora.myrac2.vip application   0/0     0/0     ONLINE    ONLINE    myrac2
```

在使用-v -t 参数组合时，无论是 State 为 ONLINE 还是 OFFLINE 的资源都会输出，其实只要是注册到 OCR 中的资源都会输出。

3.8.3 onsctl 指令

我们知道 ONS 是 Oracle Notification Service 的意思，即它提供通知服务。当服务器端发生某种事件时，就主动通知客户端事件的发生，是一种主动的事件通知机制。下面我们查看该指令的用法，然后给出示例。

例子 3-53 查看指令 onsctl 的用法。

```
[oracle@myrac2 bin]$ ./onsctl -help
Usage: ./onsctl start | stop | ping | reconfig | debug

start                    - Start opmn only.
stop                     - Stop ons daemon
ping                     - Test to see if ons daemon is running
debug                    - Display debug information for the ons daemon
reconfig                 - Reload the ons configuration
help                     - Print a short syntax description(this)
detailed                 - Print a verbose syntax description.
```

从上述输出，可以清楚地看出该指令的用法，即启动、停止 ONS 服务，或者重配置、测试

ONS 服务以及调试 ONS。

1．使用 ping 指令

下面我们先使用 ping 指令看该服务是否运行，如下例所示。

例子 3-54　测试 ONS 服务是否运行。

```
[oracle@myrac2 bin]$ ./onsctl  ping
Number of onsconfiguration retrieved, numcfg = 2
Onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
ons is running ……
```

说明 ONS 运行正常。我们注意到在上述输出中有 port 参数，而且参数值都为 6200，那么这个参数在哪里？起什么作用呢？下面我们继续分析 ONS，首先我们查看 RAC 中 ONS 的配置文件。

我们进入目录$ORACLE_HOME/opmn/conf，找到文件 ons.config，并查看该文件的内容，如下例所示。

例子 3-55　查看 ons.config 文件内容。

```
[oracle@myrac1 conf]$ pwd
/oracle/product/database/opmn/conf
[oracle@myrac1 conf]$ cat ons.config
localport=6101
remoteport=6200
loglevel=3
useocr=on
```

这里的 remoteport=6200 参数值就是使用 ping 指令中出现的 port 参数值，因为当服务器发出事件通知后，需要等待客户的响应，此时就在 6200 端口监听客户端的连接请求，完成客户对事件响应的处理过程。

loglevel 是日志记录级别，就跟踪 ONS 进程的日志信息记录级别，3 是默认值。

useocr=on 说明使用 OCR 来存储 ONS 配置信息，这个也是默认值，没必要修改。

2．使用 reconfig 指令

该指令将触发重新读取保存的 ONS 配置信息，如下例所示。

例子 3-56　重构 ONS 配置。

```
[root@myrac1 bin]# ./onsctl reconfig
```

3．停止 ONS 服务

停止 ONS 服务的指令是 onsctl stop，如下例所示。

例子 3-57　停止 ONS 服务。

```
[oracle@myrac2 bin]$ ./onsctl  stop
onsctl: shutting down ons daemon …
```

```
Number of onsconfiguration retrieved, numcfg = 2
onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
```

为了测试是否停止 ONS 服务，我们使用 ping 指令测试该服务是否还在运行，如下例所示。

例子 3-58 验证 ONS 服务是否运行。

```
[oracle@myrac2 bin]$ ./onsctl  ping
Number of onsconfiguration retrieved, numcfg = 2
Onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
ons is not running ……
```

4．启动 ONS 服务

启动 ONS 服务的指令，如下例所示。

例子 3-59 启动 ONS 服务。

```
[oracle@myrac2 bin]$ ./onsctl  ping
Number of onsconfiguration retrieved, numcfg = 2
onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
onscfg[1]
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
Number of onsconfiguration retrieved, numcfg = 2
onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
onscfg[1]
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
onsctl: ons started
```

下面测试是否成功启动 ONS 服务。

例子 3-60 测试 ONS 服务是否运行。

```
[oracle@myrac2 bin]$ ./onsctl  ping
Number of onsconfiguration retrieved, numcfg = 2
Onscfg [0]
   {node = myrac1, port = 6200 }
Adding remote host myrac1:6200
   {node = myrac2, port = 6200 }
Adding remote host myrac1:6200
ons is running ……
```

5．debug 指令

通过该指令可以查看 ONS 的状态，如监听器信息、连接信息等，如下例所示。

例子 3-61　测试 ONS 服务是否运行。

```
[oracle@myrac2 bin]$ ./onsctl  debug
Number of nosconfiguration retrieved ,numcfg = 2
Onscfg [0]
   {node = myrac1,port = 6200}
Adding remote host myrac1:6200
Onscfg[1]
  {node = myrac2 ,port = 6200 }
Adding remote host myrac2:6200
HTTP/1.1 200 OK
Content-Length:1203
Content-Type: text/html
Response:

======== ONS ========

Listeners:
 NAME    BIND  ADDRESS        PORT       FLAGS     SOCKET
----- ------------------ --------- ---------- -------
Local    127.000.000.001     6100      00000142      7
Remote  192.168.123.114     6200      00000101      8
Request       No listener

Server connections:
ID             IP        PORT       FLAGS    SENDQ    WORKER   BUSY   SUBS
----- --------------- ------- -------- -----  ------- ----- ------
    1 192.168.123.115  6200     00010005     0               1       0
Client connections:
ID             IP        PORT       FLAGS    SENDQ    WORKER   BUSY   SUBS
----- --------------- ------- -------- -----  ------- ----- ------
Pending connections:
Server connections:
ID             IP          PORT     FLAGS    SENDQ    WORKER   BUSY   SUBS
----- --------------- ------- -------- -----  ------- ----- ------
   0  127.000.000.001   6100    00020812      0               1      0
Worker Ticket: 0/0, Idle: 0

   THREAD     FLAGS
-------- ------------
153aba0     00000012
4ab1ba0     00000012
1f3bba0     00000012

Resources:

  Notification:
     Received: 0, in Receive Q: 0, Processed: 0, in Process Q: 0
Pools:
```

```
    Messages:24/25(1), Link: 25/25 (1), Subscription: 0/0 (0)
```

3.8.4 crsctl 指令

该指令的作用是监控 CRS 进程的健康状态、查看各服务的模块信息，以及维护 Voting Disk 等。该指令的参数很多，也说明它是维护 Clusterware 的重要工具。同样，我们通过帮助指令来查看该指令的用法以及相关参数。

例子 3-62 查看 crsctl 指令的用法以及参数。

```
 [root@myrac2 bin]# ./crsctl
Usage: crsctl  check   crs    -checks the viability of the CRS stack
     crsctl  check   cssd   -checks the viability of CSS
     crsctl  check   crsd   - checks the viability of CRS
     crsctl  check  evmd  - checks the viability of EVM
     crsctl  set   css<parameter> <value> - sets a parameter override
     crsctl  get   css<parameter>     -gets the value of CSS parameter
     crsctl  unset  css <parameter>     -sets CSS parameter to its default
     crsctl query   css votedisk        -lists the voting disks used by CSS
     crsctl add    css votedisk <path>   - adds a new voting disk
     crsctl delete  css votedisk <path>   - removes a voting disk
     crsctl enable  crs  -enables startup for all CRS daemons
     crsctl disable  crs  -disables startup for all CRS daemons
     crsctl start crs    - starts all CRS daemons
     crsctl stop crs   - stops all CRS daemons.Stop CRS resources in case of
cluster.
     crsctl start resource  -starts CRS resources
     crsctl stop resource  -stops CRS resources
     crsctl debug statedump evm  - dumps state info for evm objects
     crsctl debug statedump crs   - dumps state info for crs objects
     crsctl debug statedump css   - dumps state info for css objects
     crsctl debug log css  [module:level] {,module:level}……
                     -Turn on debugging for CSS
     crsctl debug trace css  -dumps CSS in-memory tracing cache
     crsctl debug log crs [module:level]{,module:level}…
                     -Turn on debugging for CRS
     crsctl debug trace crs      -dumps CRS in-memory tracing cache
     crsctl debug log evm [module:level]{,module:level}…
                     -Turns on debugging for EVM
     crsctl debug trace evm      -dumps EVM in-momory tracing cache
     crsctl debug log res <resname:level> turns on debugging for resources
     crsctl query crs softwareversion [<nodename>] - lists the version of CRS
software installed.
     crsctl query crs activeversion - lists the CRS software operating version
     crsctl lsmodules css   -lists the CSS modules that can be used for debugging
     crsctl lsmodules crs   -lists the CRS modules that can be used for debugging
     crsctl lsmodules evm  -lists the EVM modules that can be used for debugging
If necessary any of these commands can be run with additional tracing by
adding a "trace" argument at the very front.
Example: crsctl trace check css
```

显然，上面的输出已经很清楚的说明了参数的作用。下面我们给出具体的例子，并讲解输出

的含义，使读者有更直观的理解。

1．使用 check crs 参数

该参数作用是查看 crs 栈的健康状态，如下例所示。

例子 3-63 查看 crs 栈的状态。

```
[root@myrac2 bin]# ./crsctl check crs
CSS appears healthy
CRS appears healthy
EVM appears healthy
```

从输出可以看出 CRS 的三个组件运行正常，当然也可以查看单个组件的健康状况。查看组件 CRS 的健康状况如下例所示。

例子 3-64 使用 check crsd 参数查看组件 CRS 的健康状况。

```
[root@myrac2 bin]# ./crsctl check crsd
CRS appears healthy
```

同理可以通过 check cssd 指令和 evmd 指令查看组件 CSS 和组件 EVM 的健康状态。

2．使用 check css votedisk 指令查看 Voting Disk 磁盘分布

例子 3-65 查看 Voting Disk 磁盘分布。

```
[root@myrac2 bin]# ./crsctl query css votedisk
0.     0     /dev/raw/raw3
1.     0     /dev/raw/raw4
2.     0     /dev/raw/raw5

Located  3  votedisk(s)
```

3．增加和删除 Voting Disk 磁盘

以下两个维护 Voting Disk 磁盘的用法已经在 3.5 和 3.6 节给出了示例，这里就不再演示。具体指令如下例所示。

例子 3-66 添加 Voting Disk 磁盘。

```
[root@myrac2 bin]# ./crsctl add css votedisk /dev/raw/raw3
```

例子 3-67 删除 Voting Disk 磁盘。

```
[root@myrac2 bin]# ./crsctl delete css votedisk /dev/raw/raw3
```

4．使用 enable 和 disable 参数

下面两个示例说明是否在系统启动时就启动 CRS 进程，CRS 默认在系统启动时自动启动。如果不希望它自动启动，可以使用如下带 disable 参数的指令来关闭 CRS 的自动启动行为。

例子 3-68 关闭 CRS 自动启动。

```
[root@myrac2 bin]# ./crsctl disable crs
```

例子 3-69 打开 CRS 自动启动。

```
[root@myrac2 bin]# ./crsctl enable crs
```

5．使用 stop crs 和 start crs 参数

当系统运行时，出于维护的目的，需要停止当前的 CRS 服务。此时可以使用带 stop 参数的指令，而启动 CRS 服务则需要使用带 start 参数的指令，如下例所示。

例子 3-70 关闭 CRS 服务。

```
[root@myrac2 bin]# ./crsctl stop crs
Stopping resources.
Successfully stopped CRS resources
Stopping down CSS daemon
Shutdown request successfully issued
```

此时如果再使用 check crs 参数会报错，如下例所示。

```
[root@myrac2 bin]# ./crsctl check crs
Failure 1 contacting CSS daemon
Cannot communicate with CRS
Cannot communicate with EVM
```

例子 3-71 启动 CRS 服务。

```
[root@myrac2 bin]# ./crsctl start crs
The CRS stack will be started shortly
```

6．使用 lsmodules 参数

该参数的作用是列出相关 CRS 组件的模块，如列出组件 CSS 的模块，如下例所示。

例子 3-72 列出组件 CSS 的模块。

```
[root@myrac2 bin]# ./crsctl lsmodules css
The following are the CSS modules ::
CSSD
COMMCRS
COMMNS
```

例子 3-73 列出组件 EVM。

```
[root@myrac2 bin]# ./crsctl lsmodules evm
The following are the CSS modules ::
EVMD
EVMDMAIN
EVMCOMM
EVMEVT
EVMAPP
EVMAGENT
CRSOCR
CLUCLS
CSSCLNT
COMMCRS
COMMNS
```

例子 3-74 列出组件 CRS。

```
[root@myrac2 bin]# ./crsctl lsmodules crs
The following are the CSS modules ::
CRSUI
CRSCOMM
CRSRTI
CRSMAIN
CRSPLACE
CRSAPP
CRSRES
CRSCOMM
CRSOCR
CRSTIMER
CRSEVT
CRSD
CLUCLS
CSSCLNT
COMMCRS
COMMNS
```

7．查看 CRS 版本

使用 crsctl 的 query crs activeversion 来查看当前的 CRS 版本，如下例所示。

例子 3-75 查询当前的 CRS 版本。

```
[root@myrac2 bin]# ./crsctl query crs activeversion
CRS active version on the cluster is [10.2.0.1.0]
```

3.8.5 ocrcheck 指令

该指令作用检验当前的 ocr 状态信息，如当前 ocr 的设备名称、完整性检验等操作。也可以说检查 OCR（Oracle 集群注册）的健康状况。使用该指令时，会自动创建一个 log 文件，该文件位于 $ORACLE_HOME/log/<hostname>/client/ocrcheck_<pid>.log。

下面给出一个例子，然后再做给出详细的解释。

例子 3-76 执行 ocrcheck 指令。

```
[root@oracle  bin~]# ./ocrcheck
Status of Oracle Cluster Registry is as follows :
      Version                     :        2
      Total space (kbytes)        :    104336
      Used space (kbytes)         :      3824
      Available space (kbytes)    :    1000512
      ID                          :    1201248207
      Device/File Name            : /dev/raw/raw2
                                    Device / File  integrity check succeed

                                    Device / File not configured
      Cluster registry integrity check succeed
```

从上述输出可以清楚地看出整个 ocr 存储的总空间为 104336 kbytes，已经使用了 3824 kbytes，

即给出了 OCR 空间使用情况。然后是设备名，即我们配置的裸设备名为/dev/raw/raw2。

接着我们查看创建的日志信息，如下例所示。

例子 3-77 查看 ocrcheck 指令指令执行后的日志信息。

```
[root@myrac1  bin]# cd /oracle/product/database/log/myrac1/client
[root@myrac1 client]# ls -l
Total  8
-rw-r----- 1 oracle dba 730 May 7 17:27 ocr 20060 7.log
-rw-r----- 1 oracle dba 355 May 7 16:29 ocr_25248_7.log
```

3.8.6 ocrdump 指令

从该指令的名字可以猜出，它的作用是导出 OCR 的内容，但是这里的导出文件不能作为备份使用，我们可以把 OCR 的内容导出到一个本机数据文件，也可以只打印输出到屏幕，我们来看该指令的语法，如下例所示。

```
Ocrdump [<filename> | -stdout ] [-backupfile <filename>] [-keyname <keyname>]
       [-xml] [-noheader]
```

我们可以直接使用 ocrdump 指令导出 OCR 到默认的文件，这个文件名为 OCRDUMPFILE，位于当前执行指令的目录下，如下例所示。

例子 3-78 导出 OCR 的内容的默认文件。

```
[root@myrac1  bin]# ./ocrdump
```

然后，我们通过系统指令查看默认文件是否存在，如下例所示。

例子 3-79 查看默认文件 OCRDUMPFILE 是否存在。

```
[root@myrac1  bin]# ls - l
-rwxr-xr-x   1 oracle  dba   1676 May  4 19:03 cemutlo
......
......
-rw-r—r--  1 root   root   202034  May   8  10:00 OCRDUMPFILE
......
......
```



最好使用 root 用户，或者 oracle 用户必须具有在当前目录下创建文件的权限，否则会出现如下错误。

```
[oracle@myrac1  bin]# ./ocrdump
PROT-304: Failed to create dump file [OCRDUMPFILE]
```

如图 3-11 所示通过 more 指令查看的 OCRDUMPFILE 的文件内容。

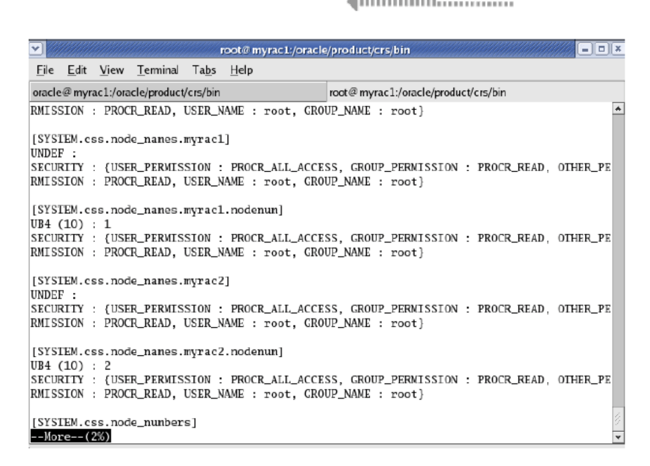


图 3-11　OCRDUMPFILE 的文件部分内容

当然也可以将 OCR 的内容直接写入指定的文件，如下例所示。

例子 3-80　将 OCR 内容写入指定文件。

```
[root@myrac1 bin]# ./ocrdump  /tmp/ocrdump
```

然后通过系统指令查看在/tmp 目录下是否成功创建该文件，如下例所示。

例子 3-81　查看是否存在 ocrdump 文件。

```
[root@myrac1 bin]# ls -l /tmp/
…….
-rw-r—r—1 root  root   202047  May  8 10:11 ocrdump
……
```

无论是直接输出到屏幕，还是输出到文件再打开查看 OCR 的内容，都会发现内容有些杂乱，但是如果使用-xml 参数，就可以更规范的显示 OCR 的内容，如图 3-12 所示。

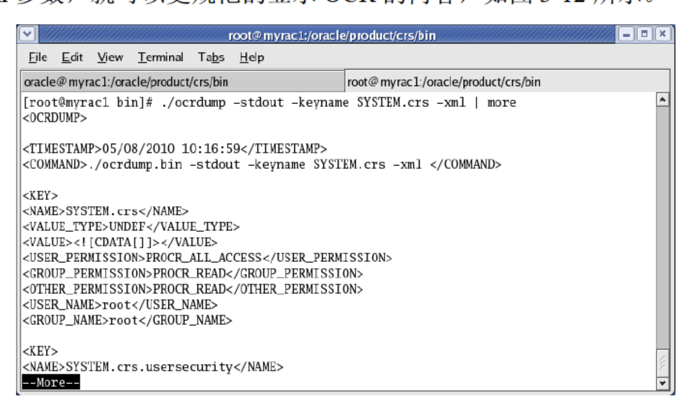


图 3-12　OCR 内容的 XML 格式输出

3.8.7　oifcfg 指令

该指令是接口配置工具，用于查看和配置 RAC 环境下各节点的网卡信息，如接口类型、物理

接口对应的 IP 地址、网卡属性等，也可以添加和删除网卡。但是这里的添加和删除网卡与操作系统级别的网卡添加与删除操作不同，它是虚拟的操作，不需要检验物理网卡是否存在。下面我们通过例子查看该指令如何使用并给出解释和示例。

例子 3-82 查看指令 oifcfg 的用法。

```
    [root@myrac1 bin]# ./oifcfg -h
    PRIF-9: incorrect usage
    Name :
           Oifcfg - Oracle Interface Configuration Tool
    Usage: oifcfg iflist [-p [-n]]
         oifcfg setif {-node <nodename> | -global} {if name/<subnet>:<if type>}…
         oifcfg getif [-node <nodename> | -global ] [ -if <if name>[/subnet]] [-type
<if type>] ]
         oifcfg delif [-noade <nodename>| -global ][<if name>[/<subnet>] ]
             oifcfg [-help]
         <nodename> - name of the host , as known to a communication network
         <if name>  - name by which the interface is configured in the system
         <subnet>   - subnet address of the interface
         <if type>  - type of the interface { cluster interconnect | public |
storage }
```

从上例输出可以看出，指令 oifcfg 有四个子操作如下所示。

- oifcfg iflist: 查看网卡接口信息（list）。
- oifcfg setif: 添加网卡接口（set）。
- oifcfg getif: 获取网卡属性。
- oifcfg delif: 删除网卡接口。

下面我们演示这些子操作的用法。

1. 使用 oifcfg iflist 指令

例子 3-83 查看网卡 ip 配置。

```
    [root@myrac1 bin]# ./oifcfg iflist -n -p
    Eth0   192.168.123.0    PRIVATE 255.255.255.0
    Eth1   10.0.0.0   PRIVATE 255.255.255.0
```

这里给出的是网卡对应的网络地址，以及子网掩码信息。

2. 使用 oifcfg getif 指令

例子 3-84 查看网卡属性。

```
    [root@myrac1 bin]# ./oifcfg  getif
    eth0   192.168.123.0    global  public
    eth1   10.0.0.0   global   cluster_interconnect
```

从上述输出中可以看出，网卡 eth0 为 public 类型，属于全局网卡类型，即网卡向外提供连接服务，也是 VIP 绑定的网卡，而网卡 eth1 类型为 cluster_interconnect ，说明该网卡提供 RAC 中节点互联的功能。

例子 3-85 查询某节点的 global 类型配置。

```
[root@myrac1 bin]# ./oifcfg getif -global -node myrac2
eth0   192.168.123.0    global  public
eth1   10.0.0.0   global   cluster_interconnect
```

这里的输出内容同上例相同，就不再解释。该例子说明可以使用-global 和-node 参数查看具体节点的详细信息，也就是“细粒度”查询。

下面我们给出更细粒度的查询，查询特定节点、特定网络接口的网卡配置信息。

例子 3-86 查询节点 myrac2 的接口 eth1 的配置信息。

```
[root@myrac1 bin]# ./oifcfg getif -if eth1 -node myrac2.0
eth1  10.0.0.0  global  cluster_inerconnect
```

3．使用 oifcfg setif 指令

该指令的作用是向 RAC 中添加一个特定的网卡配置，如误删除了一个接口配置后，可以使用该指令再次添加接口配置。

例子 3-87 添加接口配置。

```
[root@myrac1 bin]# ./oifcfg setif  -global eth0/192.168.1.0:public
[root@myrac1 bin]# ./oifcfg setif  -global eth1/10.0.0.0:cluster_interconnect
```

4．使用 oifcfg delif 指令

显然，该指令的作用就是删除一个接口配置，该指令的使用规则如下例所示。

```
oifcfg delif [-noade <nodename>| -global ][<if_name>[/<subnet>] ]
```

删除节点 myrac1 上的-global 类型接口配置如下例所示。

例子 3-88 删除节点 myrac1 上的接口配置。

```
[root@myrac1 bin]# ./oifcfg delif  -node myrac1 -global
```

3.8.8　olsnodes 指令

该指令查看集群中节点的信息，如节点名称、编号、节点的 VIP 以及网络接口名称等。下面我们通过执行该指令看看它到底有哪些功能，如下例所示。

例子 3-89 查看 olsnodes 指令的用法。

```
[root@myrac1 bin]# ./olsnodes -h
Usage: olsnodes [-n] [-p] [-i] [<node> | -l] [-g] [-v]
      Where
            -n print node number with the node name
            -p print private interconnect name with the node name
            -I print virtual IP name with the node name
            <node> print information for the local node
            -l print information for the local node
            -g turn on logging
            -v run in verbose mode
```

从上面的例子中可以看出指令 olsnodes 通过与参数的组合查看不同的内容，下面我们给出 olsnodes 的使用示例，这些示例的区别在于使用的参数不同。我们分别使用-n、-p、-i、-g、-v 参数说明 olsnodes 的使用方法。

例子 3-90 查看当前 RAC 环境下的节点。

```
[root@myrac1 bin]# ./olsnodes -n
myrac1   1
myrac2   2
```

显示当前有两个节点，第一个节点名为 myrac1，第二个节点名为 myrac2。

例子 3-91 查看所有节点的私有网络名和 VIP 网络名。

```
[root@myrac1 bin]# ./olsnodes -i -p
myrac1   myrac1-priv   myrac1-vip
myrac2   myrac2-priv   myrac2-vip
```

显示当前有两个节点，其中节点 myrac1 对应的私有网络名为 myrac1-priv，VIP 网络名为 myrac1-vip。

例子 3-92 查看某一节点的网络名。

```
[root@myrac1 bin]# ./olsnodes myrac2 -p -i
myrac2   myrac2-priv   myrac2-vip
```

显示节点 myrac2 对应的私有网络名为 myrac2-priv，VIP 网络名为 myrac2-vip。如果查看我们所操作本机上的节点信息可以采用-l 参数，如下例所示。

例子 3-93 查看本机上的节点网络名。

```
[root@myrac1 bin]# ./olsnodes -l -p -i
myrac1   myrac1-priv   myrac1-vip
```

其实不使用-l 参数可以查看整个 RAC 中节点信息（毕竟一个 RAC 中不会有上百个节点☺），但是出于周全的考虑使用-l 参数更体现灵活性的需求，如下例所示。

例子 3-94 查看节点的日志信息。

```
[root@myrac1 bin]# ./olsnodes -v
prlslms: Initializing LXL global
prlsndmain: Initializing CLSS context
prlsmemberlist: No of cluster members configured = 256
prlsmemberlist: Getting information for nodenum = 1
prlsmemberlist: node name = myrac1
prlsmemberlist: ctx->lsdata- >node num = 1
prls printdata: Printing the node data
myrac1
prlsmemberlist:Getting information for nodenum=2
prlsmemberlist: node name = myrac2
prlsmemberlist: ctx->lsdata- >node num =2
prls printdata: Printing the node data
myrac2
prlsndmain: olsnodes executed successfully
```

```
prlsdterm : Terminating LSF
```

3.9　本章小结

本章重点介绍了 Clusterware 的维护工具，OCR 和 Voting Disk 是 Clusterware 的两个重要组成部分，他们的维护是读者必须掌握的。在配置 OCR 和 Voting Disk 时最好采用 Oracle 推荐的方法，使用 Normal Redundancy 这样就实现了数据的冗余。对于管理 Clusterware 的指令，想必读者会感到很繁琐，其实，这样指令只要读者去尝试一下，并学会使用帮助，这样就很容易理解和记住指令的含义，以及参数的用法，在日常维护中只要大致知道用哪个指令即可，具体操作可以使用帮助，实际操作多了也就记住了。

第 4 章

◂RAC与Data Guard▸

Data Guard 提供数据库保护的功能，提供数据库系统高可用性以及系统可靠性的解决方案。Data Guard 是 Oracle 数据库企业版的特性，它不需要额外的软件安装，对于一般中小企业来讲可以降低软件成本，配置了 Data Guard 的数据库应用系统，在当前运行的主（Primary）库发生故障时，通过切换到备（Standby）库，就可以保证了业务的不间断性。本章我们将讲解 Data Guard 的概念、设计原理、相关名词以及如何创建物理和逻辑的备库，并详细描述主库和备库的切换过程以及如何维护备库。

4.1 Data Guard是什么

从 Data Guard 这个英文单词组合可以看出它是“数据卫士”的意思，无论你叫它数据保护或者数据卫士都不能脱离其本质，就是 Data Guard 提供了对数据库的保护，提供生产数据库的高可用性、数据保护以及灾难恢复的功能。

从数据库层次看，Data Guard 是数据库的集合，它包含一个 Primary 数据库以及一个或多个 Standby 数据库，其中 Standby 数据库就是提供 Primary 数据库保护的，一旦 Primary 数据库主机瘫痪或者数据库损坏，就可以及时切换到 Standby 数据库继续向外提供服务。

Data Guard 除了保证生产数据库的高可用性以及灾难恢复外，还能提供如数据备份以及平衡负载、平衡资源分配等任务。

4.2 Data Guard配置及相关概念

Data Guard 配置是指其组成。我们已经说过 Data Guard 由多个数据库组成，其中包括 Primary 数据库，它负责对外提供服务；还包括 Standby 数据库集合，意思是可以有不止一个 Standby 数据库，根据可靠性以及自身资源、环境需求等条件来配置。而 Primary 数据库和多个 Standby 数据库之间是通过 Oracle Net 连接。因为通过网络来连接数据库，所以对 Data Guard 中配置的数据库的物理位置就没有要求，只要网络畅通即可，如图 4-1 所示是简单的 Data Guard 配置示意图。

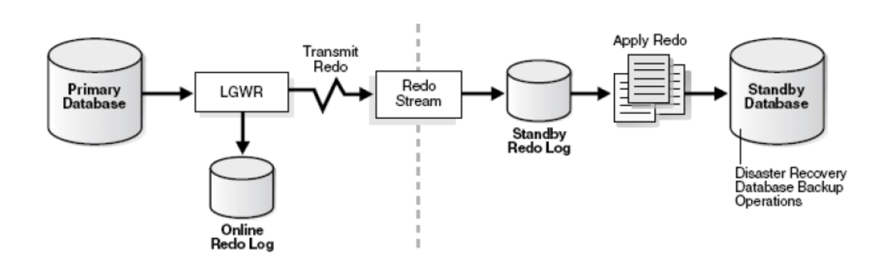


图 4-1　Data Guard 配置示意图

Primary 数据库与 Standby 数据之间通过 Oracle Net 连接，将 Primary 数据库的重做日志数据同步或异步地传输到 Standby 数据库。从上图 4-1 也可以看出 Data Guard 管理 Redo 数据流，在 Standby 数据库端通过应用 Redo 数据流到 Standby 数据库，完成与 Primary 数据库的同步。

下面我们详细介绍 Data Guard 环境涉及的三个重要概念。

1. Primary 数据库

在 Data Guard 的环境中与 Standby 数据库对应的数据库即是 Primary 数据库，也就是 Primary 数据库是正在运行的生产数据库，大多数的应用要访问该数据库，因为它在 Data Guard 环境中处于 Primary 的角色 ，所以称为 Primary 数据库。我们通过配置一个 Standby 数据库与之对应，提供数据保护以及系统的高可用性。

2. 物理 Standby 数据库

物理 Standby 数据库是 Standby 数据库的一种，物理 Standby 数据库在本质上是通过 Redo 传输服务实施 Redo 应用，将 Primary 数据库的 Redo 数据拷贝到 Standby 数据库，实现 Primary 数据库与 Standby 数据库的同步。

Standby 数据库其实就是 Primary 数据库的物理拷贝，二者的数据库结构相同。

3. 逻辑 Standby 数据库

逻辑 Standby 数据库与 Primary 数据库在物理文件组织以及数据结构方面可以不同，这点是与物理 Standby 数据库的一个区别，并且逻辑 Standby 数据库是通过 SQL 应用实现与 Primary 数据库的数据同步，SQL 应用的本质是将从 Primary 数据库获得的 Redo 数据转化成 SQL 语句，然后使用 SQL 语句实现 Redo 数据的操作。所以逻辑数据库中有自己的 Standby 本地日志文件。

逻辑 Standby 数据库可以向应用提供服务，如数据查询、报表服务等，同时不用停机就可以在逻辑 Standby 数据实现数据库软件升级以及补丁操作。

4.3　Data Guard服务本质

这里我们从本质上解释 Data Guard 如何实现数据保护。我们知道在单实例数据库环境下，通过重做日志文件用于实例恢复时保持数据的一致性，即用户提交的数据不会因为实例故障而失败。同时通过归档的重做日志文件作为在线重做日志的物理备份，这样在数据库需要介质恢复时就可以实现无数据丢失。本节我们学习 Apply 服务，它是实现数据同步的基础，然后介绍和 Standby 数据

库对应的、提供 Apply 服务的两种方法，即 Redo 应用和 SQL 应用。

4.3.1 Apply 服务

在 Data Guard 环境下，我们有一个提供主要服务的生产数据库——Primary 数据库，同时在另一个物理主机（也可以同一台主机）创建了一个数据库拷贝，我们称为 Standby 数据库。通过网络将用户的数据操作结果传输到 Standby 数据库，此时传输的数据库就是保存在生产数据库的重做日志文件中的用户操作记录数据，其实与单实例数据库保持数据一致性相同，只不过 Data Guard 通过网络来传输 Redo 数据，将 Redo 数据归档或者直接将 Redo 数据写入 Standby 数据库。这要依赖于用户在配置数据库时选择的 Redo 传输服务参数，在本章的后面部分会讲到一些 Redo 传输服务参数的含义。

传输 Redo 数据的服务成为“Apply 服务”，该服务实现了 Standby 数据库与 Primary 数据库之间的数据同步，并在第一环境下允许对这两个数据库的同时访问，这依赖于 Standby 数据库的类型。

在 Oracle Data Guard 中，将 Standby 数据库分为物理 Standby 数据库和逻辑 Standby 数据库。根据 Standby 数据库的类型，Redo 数据的 “Apply 服务”有两种实现方式，即 Redo 应用和 SQL 应用。

4.3.2 Redo 应用

Apply 服务根据创建的 Standby 数据库的不同而有差异，从本质上讲，就是实现 Redo 数据写操作的级别不同。Redo 应用服务于物理 Standby 数据库，复制 Redo 中的二进制数据。

在物理 Standby 数据库中使用 Redo 应用保持与 Primary 数据库的数据同步，Redo 应用在本质上使用介质恢复的方式来保持与 Primary 数据库的同步。即传输到 Standby 数据库的 Redo 数据块，会通过 Redo 应用直接保存在 Standby 数据库中，如图 4-2 所示。

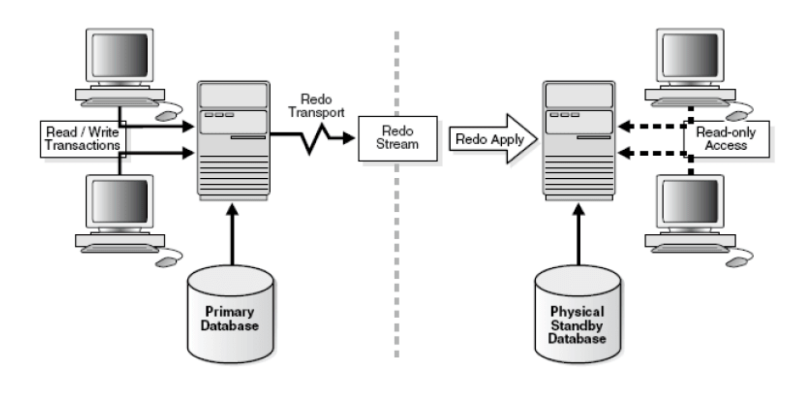


图 4-2 物理 Standby 的 Redo 应用

在图 4-2 中，Primary 数据库的 Redo 数据使用 Redo 传输服务，通过网络传输到 Standby 数据库，在 Standby 数据库上相应的进程接受 Redo 数据，并使用 Redo 应用将数据块写入数据库。此时 Standby 数据库只能用于只读访问。

4.3.3 SQL 应用

SQL 应用服务于逻辑 Standby 数据库，它将 Redo 数据中的数据转化成 SQL 语句，再执行 Redo 操作。所以 SQL 应用是一个 SQL 语句传输和重建的过程。逻辑 Standby 数据库能够以读/写模式打开，但是它维护的数据表只能以只读模式打开，用于报表查询等服务，如图 4-3 所示。

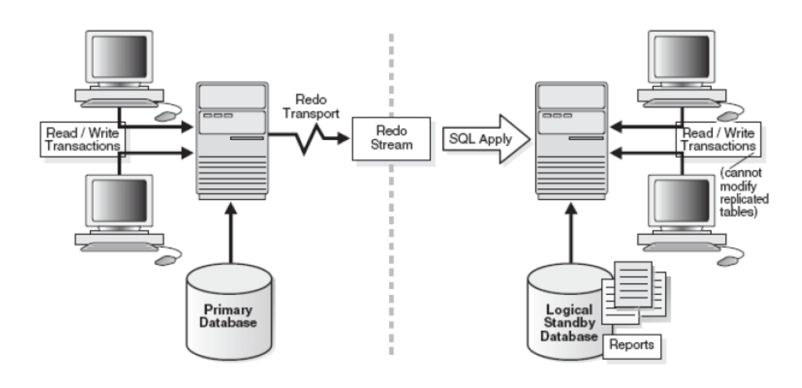


图 4-3 逻辑 Standby 的 SQL 应用

在创建逻辑 Standby 数据库时，系统会使用 SQL 应用来保存从 Primary 数据传输过来的 Redo 数据。SQL 语句在应用过程中，此时的 Standby 数据库也可以提供报表服务。

4.3.4 角色转换服务

在 Data Guard 配置中与数据库功能对应的有两种角色，即 Primary 角色和 Standby 角色。在一个 Data Guard 配置中可以有多个 Primary 数据库和多个 Standby 数据库，但是角色就两种，要么是 Primary 角色，要么是 Standby 角色。我们使用数据字典 v$database 来查询当前数据库的角色，如下例所示。

例子 4-1 查询当前数据库角色。

```
SQL> select database role
  2  from v$database;

DATABASE ROLE
----------------
PRIMARY
```

显然，上述的数据库角色为 Primary，如果是 Standby 数据库，该角色为 physical standby 或 logical standby，这要依赖于 Standby 数据库的类型。

我们可以支持如下两种角色转换，即在两种不同条件和不同需求的环境下实施的角色转换。

1．Switchover

我们可以称这种角色切换为平滑切换，是一种主动的行为，如在 Primary 数据库软件升级时可以使用 Switchover 切换将 Primary 数据库切换为 Standby 数据库，同时将 Standby 数据库切换为

Primary 数据库，当然这需要在二者的初始化参数文件中配置适当的参数，在创建 Standby 数据库时，我们会给出详细的过程及参数解释。在 Data Guard 环境下的数据库，可以使用数据字典视图 v$database 查看当前数据库可以转换为什么角色，如下例所示。

例子 4-2 查询当前数据库可以切换的角色。

```
SQL> select switchover status
  2  from v$database;

SWITCHOVER STATUS
--------------------
TO  PRIMARY
```

上述输出“TO PRIMAY”说明当前的数据库可以切换为 Primary 数据库。如果是在 Primary 数据库上查询，该参数如果是 TO STANDBY 或者 SESSION ACTIVE，则说明该数据库可以切换到 Standby 数据库。

2. Faileover

我们称 Failover 为故障发生时的切换，可以称为故障切换，即当 Primary 数据库宕机时，将启动 Standby 数据库继续对外服务。这种切换方式中，如果 Primary 数据库采用的数据保护模式是最大性能，可能会导致数据丢失；如果 Primary 数据库采用最大保护模式或者最大可用性模式，则不会造成数据丢失。模式的 Standby 数据库处于最大性能数据保护模式。接下来的一节我们讨论 Primary 数据库的数据保护模式。

4.4 Data Guard的保护模式

Data Guard 的数据保护模式是出于不同的业务或环境需求而设置的、对 Primary 数据库的保护方法，Data Guard 支持三种数据保护模式，即最高可用性（Maximum Availability）、最大性能（Maximum Performance）和最大保护（Maximum Protection）。下面我们依次介绍这三种数据保护模式的作用和相应的设置数据保护模式的指令。

1. 最高可用性（Maximum Availability）

这种数据保护模式是在保证 Primary 数据库高可用的条件下的最高级别保护模式，在 Primary 数据库中一个执行要完成两件事才会提交，一是将事务的 Redo 数据写入重做日志文件，二是将事务数据写入 Standby 数据库的重做日志文件（如果配置了多个 Standby 数据库则需要至少写入一个 Standby 数据库），而后才能提交该事务，这样该事务才会正常结束。

如果由于种种原因，如网络故障等造成 Redo 数据不能写入至少一个 Standby 数据库，则 Primary 数据库运行在最大性能模式下，实现 Redo 数据的异步传输，直到顺利将 Redo 数据同步到 Standby 数据库。这种模式不像最大保护模式会导致 Primary 数据库关闭，这也就是最高可用性的含义。

我们可以通过下例所示的方法查询当前的数据库保护模式。

例子 4-3 查询当前的数据库保护模式。

```
SQL> select protection_mode
```

```
  2  from v$database;

PROTECTION_MODE
--------------------
MAXIMUM PERFORMANCE
```

显然当前的数据保护模式为“最大性能”，这也是默认的 Primary 数据库的数据保护模式。

2．最大性能模式（Maximum Performance）

这里，最大性能的含义是在不影响 Primary 数据库性能的前提下，提供最高级别的数据保护。在 Primary 数据库端的事务，在将 Redo 数据写入在线重做日志文件后就可以顺利结束，而并不需要等待同时将 Redo 数据写入至少一个 Standby 数据库，这样使得 Primary 数据库端的事务不受过多的、与 Standby 数据库同步而造成的诸如等待事件影响，因为 Redo 数据传输是异步执行的。在数据保护方面相比最大可用性保护模式略显逊色，但是在 Primary 数据库性能上确实有提升。

最大性能数据保护模式是默认的 Data Guard 数据保护模式。

3．最大保护模式（Maximum Protection）

最大保护模式不会造成数据丢失。使用这种保护模式，在 Primary 数据库端的事务也要完成两件事才能提交，一是将事务的 Redo 数据写入重做日志文件，二是将事务数据写入 Standby 数据库（如果配置了多个 Standby 数据库则需要至少写入一个 Standby 数据库），而后才能提交该事务，也就是该事务才会正常结束。如果此时 Standby 数据库故障，无法顺利将 Redo 数据流同步到至少一个 Standby 数据库，则 Primary 数据库会宕机。

可见这种保护模式称为最大保护是名副其实的，即宁愿系统不能使用，也不会因无法保护数据而使得系统正常运行。

在 Data Guard 提供的三种数据保护方式中，我们一定要理解其分类的基础，即从对事物的不同处理方式上进行分类。最大可用性保证事务完成必须将 Redo 数据写入 Primary 库的在线 Redo 日志和 Standby 数据库，如果 Standby 库无法写入则采取异步传输方式传输 Redo 数据；最大性能的事务完成只要求将 Redo 数据写入 Primary 库的在线日志，而异步地将 Redo 数据写入 Standby 数据库；最大保护有点极端，但对于最大化数据保护也没有其他办法，即一个事务的完成需要将 Redo 数据写入 Primary 库的在线日志和 Standby 数据库，若 Standby 库无法写入则 Primary 库关闭。

4.5　Data Guard的优点

下面我们总结一下使用 Data Guard 的优点，这些优点主要是依据 Data Guard 本身的特性提取出来的。在学完本章后对这些优点会有更深入的认识和理解，这里可以先有个直观认识。

- 灾难恢复、数据保护和高可用性
- 完备的数据保护
- 高效使用系统资源
- 数据库性能平衡
- 无需额外软件
- 角色转换方便

- 易于管理

4.6 创建物理Standby数据库

通过前几节的学习，读者对 Data Guard 有了初步的了解，我们通过创建物理 Standby 数据库的过程进一步深入理解 Data Guard 的动作机制。在创建过程中会用到很多参数，我们会在创建过程中一一介绍。如果读者不看介绍而直接按照笔者的步骤创建 Standby 数据库，或许会成功，但是还是希望读者认真理解创建过程中涉及的参数含义，以及相关目录的修改。

4.6.1 创建物理 Standby 的前提条件

Data Guard 必须安装在相同的系统平台上，即 Data Guard 环境中的操作系统平台要求一致，满足了这点后在软件方面不需要额外的 Oracle 软件。因为 Data Guard 是 Oracle 企业版的一个特性，只要安装的数据库是企业版就支持创建 Data Guard。



Data Guard 环境下的数据库最好采用相同存储策略，比如 Primary 数据采用 OMF 那么 Standby 数据库也采用 OMF 文件系统。Data Guard 环境下的各数据库服务器要时间同步，不然会造成 Redo 数据传输的问题。下面我们就开始创建物理 Standby 数据库，这个过程分两个步骤进行，一个是在 Primary 数据库上的操作，一个是在 Standby 数据库上的操作。

为了读者学习方便，我们在同一台计算机上创建 Data Guard，其间的步骤和参数设置等等与在不同的计算机上设置没有两样。需要注意的步骤笔者会给出说明，使读者在不同计算机上配置 Data Guard 时更清楚每一步骤操作的含义。

4.6.2 在 Primary 数据库端的操作

在 Dataguard 环境下，对 Primary 数据库有严格的要求，比如 Primary 数据库必须处于归档模式。为了防止用户使用 nologging 子句造成无法应用对应的 Redo 数据到 Standby 数据库，最好在 database 级使用 force logging。下面是在 Primary 数据库上的必要操作步骤。

01 打开 Forced Logging 模式，如下例所示。

例子 4-4 在数据库级打开设置 force logging。

```
SQL> alter database force logging;

数据库已更改。
```

因为 Standby 数据库是通过使用 Redo 应用来实现与 Primary 数据库的数据同步的，如果在 Primary 数据库端执行 DML 或 DDL 操作时使用了 nologging 字句，会造成数据丢失。所以我们在创建 Standby 数据库之前，就使用 alter database force logging 强制使用 logging。

02 创建 Standby 数据库控制文件，如下例所示。

例子 4-5　创建 Standby 数据库控制文件。

```
SQL> alter database create standby controlfile as 'e:\backup\control01.ctl';

数据库已更改。
```

上述指令创建物理 Standby 控制文件，如果 Standby 数据库需要多个控制文件，可以继续复制创建好的控制文件并把其名字修改为参数文件中对应的名称。

03 修改参数文件。

修改参数文件是最核心的一个步骤，这里完成 Redo 传输服务的配置，以及选择 Primary 数据库的保护模式等操作。我们通过先创建 pfile 然后修改 pfile、最后再创建 spfile 的方式来修改数据库的初始化参数。首先创建 pfile，如下例所示。

例子 4-6　创建 pfile。

```
SQL> create pfile='e:\backup\initbackup.ora' from spfile;

文件已创建。
SQL> create pfile from spfile;

文件已创建。
```



上述创建的 pfile=e:\backup\initbackup.ora 参数文件用于修改 Primary 数据库端的初始化参数。而使用指令 create pfile from spfile 创建的默认目录下的初始化参数文件用于修改 Standby 数据库端参数文件。

我们修改创建的 pfile 文件 initbackup.ora，然后在该参数文件后添加如下内容。

```
db unique name=lszpri
log archive config='dg config=(lszpri,lszstdby)'
log archive dest 1='location=e:\oracle\product\10.2.0\flash recovery area\
valid for=(all logfiles,all roles) db unique name=lszpri'
log archive dest 2='service=lszstdby lgwr async
valid for=(online logfiles,primary role) db unique name=lszstdby'
log archive dest state 1=enable
log archive dest state 2=enable
remote_login_passwordfile=exclusive
```

上述参数即可以实现 Data Guard 环境中对 Primary 数据库的参数要求，但是为了完成 Switchover 角色转换，我们再增加如下内容。

```
#如下添加内容用于切换到 Standby 角色。
fal server=lszstdby
fal_client= lszpri
db file name convert='oradata\lszstdby','oradata\lszpri'
log file name convert='oradata\lszstdby','oradata\lszpri'
standby_file_management=auto
```

下面我们要解释这些参数的含义，如表 4-1 所示。

表 4-1 Primary 数据库端参数文件中增加参数的含义

参数	含义
db_unique_name=lszpri	要求primary数据库和Standby数据库端该参数名唯一，以区别每个数据库
log_archive_config='dg_config=(lszpri,lszstdby)'	要求包括Data Guard环境下所有的db_unique_name
log_archive_dest_1='location=e:\oracle\product\10.2.0\flash_recovery_area\ valid_for=(all_logfiles,all_roles) db_unique_name=lszpri'	本地归档目录，valid_for=(all_logfiles,all_roles)的含义是当前数据库运行在任何角色时所有的日志文件的本地写入目录。
log_archive_dest_2='service=lszstdby lgwr async valid_for=(online_logfiles,primary_role) db_unique_name=lszstdby'	该参数说明当前数据库角色为primary时，在线日志的Redo数据通过网络传输到db_unique_name=lszstdby的Standby数据库。Service参数说明是网络服务，lszstdby是对应Standby数据库的网络服务名，lgwr是Redo传输参数说明使用lgwr进程写Redo数据，async说明使用异步方式传输Redo数据，此时采用的是最大性能数据保护模式。Valid_for属性值说明当前数据库作为primary角色时，将online_logfiles的数据通过网络传送到远程Standby数据库。
log_archive_dest_state_1=enable log_archive_dest_state_2=enable	启动归档目录，此时这些归档目录生效。
以下参数是当switchover角色切换时使用	
fal_server=lszstdby fal_client= lszpri	设置fal_server和fal_client，在Switchover切换时fal_sever参数的数据库为primary数据库，而fal_client参数的数据库为standby数据库。
db_file_name_convert='oradata\lszstdby','oradata\lszpri' log_file_name_convert='oradata\lszstdby','oradata\lszpri'	转换数据文件和日志文件名
standby_file_management=auto	Standby数据库端的文件管理方式为AUTO，当用户对primary数据库增减或删除表空间以及数据文件等操作时，该操作会传播到Standby数据库。

接着，通过修改后的 pfile 参数文件创建 spfile 文件。首先关闭数据库，接着创建 spfile 初始化参数文件，如下例所示。

例子 4-7 创建 spfile。

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> create spfile from pfile=' e:\backup\initbackup.ora ';

文件已创建。
```

然后，启动 Primary 数据库到 open 状态，如下例所示。

例子 4-8 启动 Primary 数据库。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             167775156 bytes
Database Buffers          427819008 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
数据库已经打开。
```

04 将数据库设为归档模式。

查看数据库的归档模式，如下例所示。

例子 4-9 查看数据库的归档模式。

```
SQL> show parameter instance_name;

NAME                                 TYPE        VALUE
------------------------ --------------- ------------
instance_name                        string      lszpri
SQL> archive log list
数据库日志模式            存档模式
自动存档             启用
存档终点            e:\oracle\product\10.2.0\flash_recovery_area\
最早的联机日志序列     1
下一个存档日志序列   3
当前日志序列           3
```

输出说明已经将数据库设置为归档模式，如果没有则需要重新设置。所需的指令及步骤如下例所示。

例子 4-10 设置数据库为归档模式。

```
SQL>shutdown immediate;
SQL>startup mount;
SQL>aler database archivelog;
SQL>alter database open;
```

05 配置监听以及网络服务。

使用 netca 工具配置监听以及创建网络服务，网络服务名分别为 lszpri 和 lszstdby。我们创建监听后，重启监听，如下例所示。

例子 4-11 关闭和启动监听。

```
E:\oracle\product\10.2.0\db_1\BIN>lsnrctl stop
E:\oracle\product\10.2.0\db_1\BIN>lsnrctl start
```

创建服务名时的两个重要参数如图 4-4、图 4-5 所示。



图 4-4　配置网络服务名



图 4-5　配置主机地址和端口

如上图所示，我们创建两个服务名一个是 lszpri，一个是 lszstdby，两个服务对应的主机地址和端口要相同，因为我们是在本机上做一个模拟试验，所以设置相同。



如果 Standby 数据库是安装在不同的主机上，只要在 Standby 数据库所在的计算机上启用 netca 工具，配置 lszpri 网络服务名称时，将主机名参数改为对应的 ip 地址即可。因为 Standby 数据库要与 Primary 数据库通信，所以需要在 Standby 数据库端设置代表 Primary 数据库的 lszpri 服务名并做测试。

在创建完成网络服务名后，我们使用 tnsping 指令测试其运行是否正常，如下例所示。

例子 4-12 测试网络服务名 lszpri 是否正常。

```
    C:\Documents and Settings\Administrator>tnsping lszpri

    TNS Ping Utility for 32-bit Windows: Version 10.2.0.1.0 - Production on 11-5
月 -2010 09:57:00

    Copyright (c) 1997, 2005, Oracle.  All rights reserved.

    已使用的参数文件:
    e:\oracle\product\10.2.0\db_1\network\admin\sqlnet.ora

    已使用 TNSNAMES 适配器来解析别名
    Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL =
TCP)(HOST = 127.0.0.1)(PORT = 1521))) (CONNE
    CT_DATA = (SERVICE_NAME = lszpri)))
    OK (30 毫秒)
```

例子 4-13 测试网络服务名 lszstdby 是否正常。

```
    C:\Documents and Settings\Administrator>tnsping lszstdby

    TNS Ping Utility for 32-bit Windows: Version 10.2.0.1.0 - Production on 11-5
月 -2010 09:57:04

    Copyright (c) 1997, 2005, Oracle.  All rights reserved.
```

```
已使用的参数文件:
e:\oracle\product\10.2.0\db_1\network\admin\sqlnet.ora


已使用 TNSNAMES 适配器来解析别名
Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1521))) (CONNE
CT DATA = (SERVICE NAME = lszstdby)))
OK (30 毫秒)
```



因为 Primary 数据库要和 Standby 数据库通过网络服务来传输 Redo 数据，所以此时在 Primary 数据库上要创建与 Standby 数据库通信的 lszstdby 服务名。该网络服务通过 netca 工具在 Primary 数据库端实现。

06 备份 Primary 数据库的参数文件，作为 Standby 数据库的参数文件修改模板，如下例所示。

例子 4-14 备份 Primary 数据库参数文件。

```
SQL> host copy e:\oracle\product\10.2.0\db 1\database\initlszpri.ora e:\backup\initlszstdby.ora
已复制         1 个文件。
```

07 备份数据文件、Standby 控制文件以及参数文件到 Standby 数据库。

该步骤的完成方式很多，总之需要将我们创建的 Standby 数据库控制文件、Primary 数据库的数据文件、重做日志文件、备份的 pfile 参数文件拷贝到 Standby 数据库端。

参数文件的位置没有什么限制，它的目的是在 Standby 数据库端修改后创建 spfile 文件。

而对于数据文件以及重做日志文件、控制文件要拷贝到与 Standby 数据库端相应参数对应的目录中。此时可以先将这些数据拷贝到 Standby 数据库，然后在设置完初始化参数后再把数据文件和控制文件拷贝到相对应的目录下。

4.6.3 创建物理 Standby 数据库

在上节中，我们完成了 Primary 数据库端的准备工作，现在转移到 Standby 数据库创建可用的 Standby 数据库，操作步骤如下。

01 创建新的 Oracle Service，在 Standby 数据库端创建新的实例。

例子 4-15 在 Standby 数据库端创建新的实例。

```
D:\>oradim -new -sid lszstdby -startmode manual
实例已创建。
```

02 创建密码文件，如下例所示。

例子 4-16 创建密码文件。

```
D:\>orapwd     file=e:\oracle\product\10.2.0\db_1\database\PWDlszstdby.ora password=oracle entries=30
```



此时创建的密码文件中的密码必须和 Primary 数据库的 sys 用户相同。因为在 Redo 传输时使用该用户确认连接。

03 创建存储数据文件的目录，以及其他相关目录。

因为 Standby 数据库在被切换到 Primary 数据库时，它就是一个完备的数据库系统，所以在正常创建数据库时所建立的目录，这里都需要重新根据实际情况创建。如这里我们使用全局服务名为 lszstdby，则在 oradata 下创建该目录。

例子 4-17 在 oradata 下创建目录。

```
mkdir e:\oracle\product\10.2.0\oradata\lszstdby
mkdir e:\oracle\product\10.2.0\admin\lszstdby\udump
…
```

针对 Primary 数据库的目录结构去创建 Standby 数据库所需的目录结构。之后，将从 Primary 数据库备份到 Standby 数据库端的文件拷贝到相应目录，如数据文件和控制文件要拷贝到目录 E:\oracle\product\10.2.0\oradata\lszstdby 下。并且控制文件的数量以及控制文件名都要和下一步创建初始化参数文件中相应的参数内容相一致。

04 创建初始化参数文件。

我们在 Primary 数据库中已经备份了初始化参数文件，并且已经将该文件拷贝到 Standby 数据库所在主机。我们下面需要修改该文件，并添加一些内容。

首先向备份的参数文件中增加如下内容。

```
db_unique_name=lszstdby
log archive config='dg config=(lszpri,lszstdby)'
db_file_name_convert='oradata\lszpri','oradata\lszstdby'
log_file_name_convert='oradata\lszpri','oradata\lszstdby'
log_archive_format=log%t_%s_%r.arc
log_archive_dest_1='location= E:\oracle\product\10.2.0\oradata\lszstdby
valid_for=(all_logfiles,all_roles) db_unique_name=lszstdby'
log_archive_dest_state_1=enable
#如下添加内容用于切换到 primary 角色。
log_archive_dest_2='service=lszpri lgwr async
valid_for=(online_logfiles,primary_role) db_unique_name=lszpri'
log_archive_dest_state_2=enable
remote_login_passwordfile=exclusive
fal_server=lszpri
fal_client=lszstdby
standby_file_management=auto
```



上述参数的含义与 Primary 数据库端参数文件中增加的参数含义相同，只是参数值不同。

接着，我们还需要修改所有与 lszpri 相关的目录为当前 lszstdby 数据库的目录，如修改控制文件的内容如下。

```
control_files='e:\oracle\product\10.2.0/oradata/lszstdby/\control01.ctl',
```

```
'e:\oracle\product\10.2.0/oradata/lszstdby/\control02.ctl',
'e:\oracle\product\10.2.0/oradata/lszstdby/\control 03.ctl'
```



该步骤虽然简单，但是容易出错，希望读者细心阅读参数文件的每一项内容，针对 lszstdby 数据库文件目录的实际情况做出修改。

创建 Standby 数据库的初始化参数文件，首先切换 Oracle Service 到 lszstdby，如下例所示。

例子 4-18 切换 Oracle Service 到 lszstdby 并连接实例。

```
D:\>set oracle sid=lszstdby

D:\>sqlplus /nolog

SQL*Plus: Release 10.2.0.1.0 - Production on 星期二 5月 11 10:35:21 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> conn /as sysdba
已连接到空闲例程。
```

接着创建服务器初始化参数文件。

```
SQL> create spfile from pfile='e:\backup\initbackup.ora';

文件已创建。
```

在创建完成初始化参数文件后，需要注意，此时必须将数据文件和控制文件拷贝到参数文件中指定的目录下，并注意修改文件名称。

05 启动 Standby 到 mount 状态，如下例所示。

例子 4-19 启动 Standby 数据库到 mount 状态。

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             167775156 bytes
Database Buffers          427819008 bytes
Redo Buffers                7135232 bytes

数据库已更改。
```

上述操作结果说明已经成功打开 Standby 控制文件。下面我们验证看看当前的数据库实例是否为 lszstdby。

例子 4-20 验证当期的数据库实例。

```
SQL> show parameter instance_name

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------
```

```
instance_name                         string      lszstdby
```

输出说明当前的数据库实例为我们新创建的 Standby 数据库实例 lszstdby。

06 启动 redo 应用。

我们知道 redo 应用是物理 Standby 数据库实现与 Primary 库“同步”的方式，下面说明如何在 Standby 数据库端启动 Redo 传输服务。

例子 4-21 启动 redo 服务。

```
SQL> alter database recover managed standby database disconnect from session;

数据库已更改。
```

这里 disconnect from session 选项的作用是使得 Redo 在后台会话进行，但是希望读者省略该参数，这样等待成功启动 Redo 传输服务后再进行其他操作。

07 验证 Standby 数据库是否正常。

这里正常的含义是 Standby 数据库是否可以接收了 Primary 库的归档日志。我们需要分步完成以下操作：

（1）在 Primary 数据库上查询当前的归档的日志文件。

例子 4-22 查询已经归档的日志序列号以及归档时间。

```
SQL> select sequence#,first time,next time
  2  from v$archived log
  3  order by sequence#;

 SEQUENCE# FIRST TIME      NEXT TIME
---------- -------------- --------------
         2 12-5 月 -10     12-5 月 -10
```

此时，输出说明当前的已经归档的日志序列号为 2，因为是采用异步传输方式，所以我们强制 Primary 数据库端切换日志文件，从而使得 Standby 数据库端产生新的归档文件。

（2）在 Primary 库上人工切换日志文件。

例子 4-23 Primary 库上切换日志文件。

```
SQL> alter system switch logfile;

系统已更改。
```

一旦在 Primary 数据库端强制切换日志，Standby 数据库端也会切换日志文件，并归档接收到的 Redo 数据。

（3）在 Standby 数据库验证是否有新的归档文件。

例子 4-24 验证是否有新的归档文件。

```
SQL>  select sequence#,first time,next time
  2   from v$archived log
  3   order by sequence#;

 SEQUENCE# FIRST_TIME      NEXT_TIME
```

```
---------- -------------- --------------
       2 12-5月 -10     12-5月 -10
```

上述操作在 Standby 数据库端查询到信息，说明在 Primary 数据库端的归档的日志文件数据已经传输到了 Standby 数据库。而且二者的序列号相同，说明是同一个归档的日志文件。

通过上述的演示说明，在 Primary 数据库端的日志数据流可以传输到 Standby 数据库，此时我们已经成功创建了包含一个物理 Standby 数据库和一个 Primary 数据库的 Data Guard。

我们通过数据字典 v$database 来查看 Standby 数据库的状态信息，如下例所示。

例子 4-25　查看 Standby 数据库的状态信息。

```
SQL> select db unique name,switchover status,database role
  2  from v$database;

DB UNIQUE NAME       SWITCHOVER STATUS    DATABASE ROLE
---------------- -------------------- ----------------
lszstdby             TO PRIMARY           PHYSICAL STANDBY
```

我们看到数据库 lszstdby 的数据库角色 DATABASE_ROLE 为 PHYSICAL STANDBY。而 SWITCHOVER_STATUS 状态为 TO PRIMARY，说明此时的 Standby 数据库可以切换为 Primary 数据库。

4.7　Standby的角色转换

Oracle Data Guard 支持两种情况下的角色转换，一个是 Switchover 角色转换，一个是 Failover 角色转换。无论是逻辑 Standby 数据库还是物理 Standby 数据库都支持这两种角色转换。

4.7.1　物理 Standby 的 Switchover

物理 Standby 的 Switchover 是在物理 Standby 的环境下实现平滑的数据库切换，将 Primary 数据库切换为 Standby 数据库，而同时将 Standby 数据库切换为 Primary 数据库向外部提供主要的数据库服务。此时切换后成为 Standby 的数据库可以进行软件升级以及其他测试任务。这种角色切换不会造成数据丢失，具体步骤如下。

01 检查 Primary 数据库是否支持 Switchover 操作。

这个步骤是必须的，因为如果 Primary 数据库端的设计 Redo 传输以及角色转换的参数设置不正确会导致不允许做角色转换。我们通过下例查询当前数据库的 Switchover 状态。

例子 4-26　查询当前数据库的 Switchover 状态。

```
SQL> select switchover status from v$database;

SWITCHOVER STATUS
--------------------
TO STANDBY
```

上述输出为 TO STANDBY，说明可以切换为 Standby 数据库，如果上述参数为 SESSION

ACTIVE 读者也不必紧张，此时因为有会话存在的原因，它也表示可以切换为 Standby 数据库。

02 在 Primary 数据库启动 Swtichover。

在 Primary 数据库端进行 Switchover 角色转换，注意我们使用的指令。

例子 4-27 Switchover 角色转换。

```
SQL> alter database commit to switchover to physical standby;

数据库已更改。
```

上述语句将 Primary 数据库切换为 Standby 数据库，并且，当前的控制文件备份到 SQL 会话跟踪文件中，对于重建控制文件提供了依据。



如果在查询 switchover_status 时，显示的值为 SESSIONS ACTIVE，则需要在上述语句后添加 WITH SESSION SHUTDOWN 语句，即在切换到 Standby 数据库时先关闭当前会话。

03 启动 Primary 数据库到 mount 状态，如下例所示。

例子 4-28 启动数据库到 mount 状态。

```
SQL> shutdown immediate;
ORA-01507: 未装载数据库

ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             171969460 bytes
Database Buffers          423624704 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
```

此时原来的 Primary 数据库已经是 Standby 数据库了。

04 切换到 Standby 数据库，如下例所示。

例子 4-29 切换到 Standby 数据库。

```
D:\>set oracle_sid=lszstdby

D:\>sqlplus /nolog

SQL*Plus: Release 10.2.0.1.0 - Production on 星期二 5月 11 14:08:42 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> conn /as sysdba
已连接。
```

检查该数据库的 Switchover 状态，看是否可以切换为 Primary 数据库，以防不测，如下例所示。

例子 4-30 检查当前 Standby 数据库的状态。

```
SQL> select switchover status
  2  from v$database;

SWITCHOVER STATUS
--------------------
TO PRIMARY
```

显然 switchover_status 的状态为 TO PRIMARY，该参数值说明当前的 Standby 数据库可以切换为 Primary 数据库。

05 将 Standby 数据库转换为 Primary 角色，如下例所示。

例子 4-31 转换成 Primary 角色。

```
SQL> alter database commit to switchover to primary;

数据库已更改。
```

06 将此时的 Primary 数据库切换到 OPEN 状态，如下例所示。

例子 4-32 把 Primary 数据库切换到 OPEN 状态。

```
SQL> alter database open;

数据库已更改。
```

此时，通过控制文件打开了当前（原来的 Standby 数据库）Primary 数据库的数据文件，并可以对外提供服务了。

07 查看当前的数据文件。

我们查看转换后的 Primary 数据库的数据文件分布情况，如下例所示。

例子 4-33 查看 Primary 数据库的数据文件。

```
SQL> select name
  2  from v$datafile;

NAME
------------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF
```

我们看到当前的数据文件已经切换到了 oradata\lszstdby 目录下。那控制文件是怎么知道这个目录的呢。之前，我们在 Primary 数据库的参数文件中添加了如下所示的内容。

```
db_file_name_convert='oradata\lszpri','oradata\lszstdby'
```

就是这个参数使得控制文件知道 Standby 数据库中数据文件的位置。读者这里应该理解了参数文件中参数 db_file_name_convert 的实际含义了。

现在，我们再看一下当前的数据库实例名，如下例所示。

例子 4-34 查看当前数据库实例名。

```
SQL> show parameter instance name;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------
instance_name                        string      lszstdby
```

显然当前 Primary 数据库实例名为 lszstdby。接下来我们通过日志数据流是否成功传输来确定当前 Switchover 切换是否成功。

08 通过 Primary 数据库切换日志查看 Standby 数据库是否正常。

首先，查看当前 Primary 数据库的重做日志，如下例所示。

例子 4-35 查 Primary 数据库的重做日志。

```
SQL> select sequence#,first_time,next_time
  2  from v$archived_log
  3  order by sequence#;

 SEQUENCE# FIRST_TIME     NEXT_TIME
---------- -------------- --------------
         3 11-5月 -10     11-5月 -10
         4 11-5月 -10     11-5月 -10
         5 11-5月 -10     11-5月 -10
         6 11-5月 -10     11-5月 -10
         7 11-5月 -10     11-5月 -10
         8 11-5月 -10     11-5月 -10
         9 11-5月 -10     11-5月 -10

已选择 8 行。
```

显然，上述归档的重做日志的最大序列号为 9，我们通过在 Primary 数据库端人工切换重做日志文件来触发 Redo 传输，如下例所示。

例子 4-36 在 Primary 数据库端切换日志。

```
SQL> alter system switch logfile;

系统已更改。
```

接着，我们查看 Standby 数据库的归档日志信息，如下例所示。

例子 4-37 查看 Standby 数据库的归档日志信息。

```
SQL> select sequence#,first_time,next_time
  2  from v$archived_log
  3  order by sequence#;

 SEQUENCE# FIRST_TIME     NEXT_TIME
---------- -------------- --------------
         2 11-5月 -10     11-5月 -10
         3 11-5月 -10     11-5月 -10
         4 11-5月 -10     11-5月 -10
```

```
          5 11-5 月 -10      11-5 月 -10
          6 11-5 月 -10      11-5 月 -10
          7 11-5 月 -10      11-5 月 -10
          8 11-5 月 -10      11-5 月 -10
          9 11-5 月 -10      11-5 月 -10
         10 11-5 月 -10      11-5 月 -10

已选择 15 行。
```

显然上述输出中增加了一个归档日志文件，序列号为 10，此时说明数据库角色转换成功。

4.7.2　物理 Standby 的 Failover

物理 Standby 的 Failover 是当 Primary 数据库不能使用时，比如 Primary 数据库宕机，此时实现 Failover 使得 Standby 数据库可以继续向用户提供服务。不过需要说明，如果用户端通过配置 Oracle Net 与数据库服务器通信，需要修改 tnsnames.ora 文件，修改诸如数据库服务器的主机名、协议及端口等内容，这依赖于 Standby 数据库的配置而定。下面我们演示如何实现物理 Standby 的 Failover。

现在我们当前的 Standby 数据库为 lszpri，而 Primary 数据库为 lszstdby。我们假设当前的 Primary 数据库故障，无法继续工作，需要切换到 Standby 数据库 lszpri，使其继续对用户服务。具体步骤如下。

01 检查数据库 lszpri 上的归档日志文件是否连续，如下例所示。

例子 4-38 检查归档日志是否连续。

```
SQL> select thread#,low sequence#,high sequence#
  2  from v$archive gap;

未选定行
```

如果输出结果是“未选定行”，说明在 Standby 数据库上的归档日志是连续的不连续的归档日志是无法使用的，比如缺少序列号为 60 和 61 的归档日志，则 61 序号后的归档日志都无法使用，这样会造成数据丢失，如下例所示。

例子 4-39 检查归档日志文件是否连续。

```
SQL> select thread#,low_sequence#,high_sequence#
  2  from v$archive_gap;
TREAD#    LOW SEQUENCE#   HIGH SEQUENCE#
-------- -------------- ----------------
    1                60               61
```

上述输出说明缺少序列号为 60、61 的重做日志文件，此时必须从 Primary 数据库拷贝这些缺少的重做日志文件到 Standby 数据库，并使用如下方式注册拷贝过来的日志文件。

```
SQL> ALTER DATABASE REGISTER HPYSICAL LOGFILE 'filespec1';
```

在注册完这些缺少的重做日志文件后，Standby 数据库会自动进行归档并应用这些 Redo 数据。

02 检查日志文件是否完整。

查看当前两个数据库中已经归档的日志文件的最大序列号是否相同。在两个数据库上执行如下指令。

在 Standby 数据库上的执行，如下例所示。

例子 4-40 查询 Standby 数据库已经归档的最大日志序列号。

```
    SQL> select unique thread# as thread ,max(sequence#) over (partition by thread#)
as last
      2  from v$archived_log;

       THREAD       LAST
    ---------- ----------
             1         10
```

如果 Standby 数据库上的日志文件不连续，则会出现一个 thread 有多个最大序列号，如下例所示。

例子 4-41 查询当前已经归档的最大日志序列号。

```
    SQL> select unique thread# as thread ,max(sequence#) over (partition by thread#)
as last
      2  from v$archived log;

       THREAD       LAST
    ---------- ----------
             1          6
             1         10
```

此时，说明 Standby 数据库的日志不完整，需要拷贝序列号为 7、8、9 的日志文件，并且使用如下指令来注册。

```
    SQL> ALTER DATABASE REGISTER HPYSICAL LOGFILE 'filespec1';
```

在 Primary 数据库执行如下例所示的指令。

例子 4-42 在 Primary 数据库查询当前的最大归档的日志序列号。

```
    SQL> select unique thread# as thread ,max(sequence#) over (partition by thread#)
as last
      2  from v$archived_log;

       THREAD       LAST
    ---------- ----------
             1         10
```

通过查询，我们知道二者的重做日志最大序列号相同，都为 10。所以 Standby 数据库的日志文件完整。

03 停止 Standby 数据上的 Redo 应用，如下例所示。

例子 4-43 停止 Redo 应用。

```
    SQL> alter database recover managed standby database cancel;
```

```
数据库已更改。
```

04 完成所有归档日志的应用并启动 Failover，如下例所示。

例子 4-44 完成所有归档日志的应用并启动 Failover。

```
SQL> alter database recover managed standby database finish force;

数据库已更改。
```

该语句的目的是将所有收到的 Redo 数据写入 Standby 数据库，如果此时出现错误使得 Redo 数据无法写入数据库，而且问题无法解决，可以使用如下语句继续完成 Failover 的切换。

```
SQL> ALTER DATABAE ACTIVE PHYSICAL STANDBY DATABASE
```

05 验证当前 Standby 数据库是否可以切换为 Primary 数据库，如下例所示。

例子 4-45 查询当前 Standby 数据库的 SWITCHOVER 状态。

```
SQL> select switchover_status
  2  from v$database;

SWITCHOVER_STATUS
--------------------
TO PRIMARY
```

SWITCHOVER_STATUS 的状态为 TO PRIMARY，说明可以切换为 Primary 数据库。

06 把物理 Standby 数据库切换为 Primary 角色，如下例所示。

例子 4-46 把物理 Standby 数据库切换为 Primary 角色。

```
SQL> alter database commit to switchover to primary ;

数据库已更改。
```

07 打开数据库，如下例所示。

例子 4-47 打开数据库。

```
SQL> alter database open;

数据库已更改。
```



此时的 Primary 数据库是 lszpri，但是数据库 lszstdby 已经不是 Data Guard 的一部分了，所以此时为了系统的可靠性，最好重建一个 Standby 数据库。

4.8 管理物理Standby数据库

Standby 数据库创建完成后，其维护就成为 DBA 的一个重要任务，这样在 Primary 数据库发生故障时就可以轻松地切换到 Standby 数据库了。本节我们讲解以各种模式打开 Standby 数据库，Standby 数据库的文件管理以及管理 Standby 数据库的数据字典视图。

4.8.1 启动 Standby 数据库

首先查看 Standby 数据库当前状态是 Redo 应用状态还是处于 OPEN 状态，如下例所示。

例子 4-48 查询 Standby 数据库的当前状态。

```
SQL> select status from v$instance;

STATUS
------------
MOUNTED
```

从这个例子看出，当前的 Standby 数据库处于 Redo 应用状态，此时如果打开数据库必须先取消 Redo 应用，如下例所示。

例子 4-49 取消物理 Standby 的 Redo 应用。

```
SQL> alter database recover managed standby database cancel;

数据库已更改。
```

然后启动 Standby 数据库，此时我们不使用任何参数，默认启动到 read-only 模式，如下例所示。

例子 4-50 启动 Standby 数据库。

```
SQL> alter database open;

数据库已更改。
```

下面，我们再将 Standby 数据库切换到 Redo 应用模式，如下例所示。

例子 4-51 启动 redo 应用。

```
SQL>  alter database recover managed standby database disconnect from session;

数据库已更改。
```

此时 Standby 数据库会自动切换到 MOUNT 状态，如下例所示。

例子 4-52 查询 Standby 数据库的当前状态。

```
SQL> select status
  2  from v$instance;

STATUS
------------
MOUNTED
```



如果数据库已经处于 SHUTDOWN 状态，可以直接在 SQLPLUS 中使用 startup 来打开 Standby 数据库。

4.8.2　关闭 Standby 数据库

要关闭 Standby 数据库只需要使用 shutdown immediate 指令，该指令会导致 Standby 数据库首先停止 Redo 应用，然后关闭 Standby 数据库，如下例所示。

例子 4-53　关闭 Standby 数据库。

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```



如果 Primary 数据库正在运行且延迟传输 Redo 数据到 Standby 数据库，即在参数 log_archive_dest_n 中设置了 delay 参数，此时需要先在 Primary 数据库上执行一次日志切换，然后再关闭 Standby 数据库。

4.8.3　Primary 数据库结构变化的传播

这里的数据库结构变化是指在 Primary 数据库端增减或删除数据文件、表空间、重命名文件，以及修改表空间的状态等操作，这些操作是否可以自动传播到 Standby 数据库，取决于参数 STANDBY_FILE_MANAGEMENT 的设置。如果该参数设置为 AUTO，则上述的数据库变化会反映到 Standby 数据库；如果该参数设置为 MANUAL，则需要手工在 Standby 数据库做修改。

4.8.4　自动传播数据文件和表空间的变化

在向 Primary 数据库添加数据文件和表空间时，该行为是否可以传播到 Standby 数据库取决于 Primary 数据库端参数 STANDBY_FILE_MANAGEMENT 的设置。如果该参数设置为 AUTO，则对数据文件和表空间的操作会传播到 Standby 数据库。

现在我们给出一个综合示例来验证这个传播过程，在 Primary 数据库的 USERS 表空间添加一个数据文件 test.dbf，然后在 Standby 数据库查看该数据文件是否被同步创建。

01 查看 Primary 数据库的数据文件，如下例所示。

例子 4-54　查询当前 Primary 数据库的数据文件。

```
SQL> select name
  2  from v$datafile;

NAME
---------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF
```

注意，此时输出的数据文件名没有 test.dbf 文件，下面我们要在 Primary 数据库端创建该数据

文件并添加到 USERS 表空间。

02 向 Primary 数据库添加数据文件，如下例所示。

例子 4-55 向 USERS 表空间增加数据文件。

```
SQL> alter tablespace users
  2  add datafile 'E:\ORACLE\PRODUCT\10.2.0\oradata\lszpri\test.dbf' size
100m;

表空间已更改。
```

03 在 Primary 数据库查看是否成功添加数据文件。

我们在第二步已经在 Primary 数据库上的 USERS 表空间添加了一个数据文件 test.dbf。下面我们再次查看数据文件名，如下例所示。

例子 4-56 验证数据文件添加结构。

```
SQL> select name
  2  from v$datafile;

NAME
---------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\TEST.DBF
```

我们看到数据文件 test.dbf 添加成功。下面我们转移到 Standby 数据库查看该文件是否被成功创建。

04 在 Primary 数据库切换日志文件，如下例所示。

例子 4-57 切换日志文件。

```
SQL> alter system switch logfile;

系统已更改。
```

这个步骤的目的是实现 Redo 应用，这样 Primary 数据库端的数据文件操作就可以传播到 Standby 数据库。下面我们转移到 Standby 数据库。

05 在 Standby 数据库查看数据文件是否成功添加，如下例所示。

例子 4-58 查看 Standby 数据库数据文件。

```
SQL> select name
  2  from v$datafile;

NAME
---------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSAUX01.DBF
```

```
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\TEST.DBF
```

从 Standby 数据库端的输出可以看出，增加的数据文件会自动添加到 Standby 数据库中。这里如果是删除操作也会同样传播到 Standby 数据库，读者可以自己验证。

4.8.5　手工修改数据文件和表空间的变化

当参数 STANDBY_FILE_MANAGEMENT=MANUAL 时，在 Primary 数据库端数据文件和表空间的变化（删除、增减和修改）不会自动传播到远端的 Standby 数据库，这时候就需要 DBA 手工修改，以反映 Primary 数据库端的结构变化。

本节我们在 Primary 数据库端创建一个表空间，验证表空间的变化是否可以传播到 Standby 数据库。我们先修改参数 STANDBY_FILE_MANAGEMENT 为 MANUAL，如下例所示。

例子 4-59　修改参数 STANDBY_FILE_MANAGEMENT。

```
SQL> alter system set standby file management=manual;

系统已更改。
```

下面我们在 Primary 数据库上创建表空间 testtbs 看该表空间的创建是否可以传播到 Standby 数据库，以及如何在 Standby 数据库手工同步这种变化。

01 在 Primary 数据库创建表空间，如下例所示。

例子 4-60　创建表空间。

```
SQL> create tablespace testtbs datafile 'e:\oracle\product\10.2.0\oradata\
lszpri\tbstest.dbf;

表空间已创建。
```

接着为了验证 Primary 数据库端的表空间创建结果，我们查询数据文件名，如下例所示。

例子 4-61　查询数据文件名。

```
SQL> select name
  2  from v$datafile;

NAME
---------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\TBSTEST.DBF
```

我们再次查询当前的表空间名，如下例所示。

例子 4-62　查询表空间名。

```
SQL> select tablespace_name
  2  from dba_tablespaces;
```

```
TABLESPACE_NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
MYTEST
```

上述的查询结果说明我们已经成功创建了表空间，我们接着执行日志切换。

```
SQL> alter system switch logfile;

系统已更改。
```

下面我们切换到 Standby 数据库，看 Primary 数据库端的操作是否传播过去。

02 在 Standby 数据库端查询数据文件和表空间。

例子 4-63 查询数据文件。

```
SQL> select name
  2  from v$datafile;

NAME
--------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF
E:\ORACLE\PRODUCT\10.2.0\DB_1\DATABASE\UNNAMED00009
```

注意此时在 Standby 数据库的数据文件中增加了以前没有的文件名，显然这个事件的发生和我们在 Primary 数据库端创建表空间有关。我们再继续查看表空间文件名。

例子 4-64 在 Standby 数据库端查询表空间名。

```
SQL> select tablespace_name
  2  from dba_tablespaces;

TABLESPACE_NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
MYTEST
```

我们看到表空间名可以自动在 Standby 数据库端创建，那么如何把 Primary 数据库端表空间创建的物理变化，即数据文件的位置映射给 Standby 数据库呢，我们使用如下指令实现。

例子 4-65 在 Standby 数据库端手工修改数据文件名。

```
SQL> alter database create datafile
'E:\ORACLE\PRODUCT\10.2.0\DB 1\DATABASE\UNNAMED00007'
As 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\TBSTEST.DBF';

数据库已修改。
```

在上述命名完成后，只是在数据字典和控制文件中更改了文件记录，但是在新的文件目录下并没有实际的物理文件，所以此时需要将文件 UNNAMED00007 拷贝到新的文件目录，并修改名称为 TBSTEST.DBF。

如果此时在 Primary 数据库上执行删除表空间操作，如下列所示。

例子 4-66 删除表空间及数据文件。

```
SQL> drop tablespace tbstest including contents and datafiles

表空间已删除。
```

此时在 Primary 数据库上会自动删除表空间定义以及对应的数据文件，在 Standby 数据库上也会删除表空间以及数据文件定义，即从数据字典和控制文件中删除这些记录，但是和表空间对应的物理文件还需要手工删除。其实这个过程和在 Standby 数据库端添加数据文件的操作相反。

4.8.6 重命名数据文件

在 Data Guard 环境下，重命名文件的这种变化与参数 STANDBY_FILE_MANAGEMENT 无关，无论该参数是 AUTO 或 MANUAL 都需要 DBA 手工在 Standby 数据库操作来实现结构的同步。我们通过一个例子演示这个重命名文件的过程。

1. 在 Primary 数据库端

要修改文件名，首先将表空间 USERS 的状态设置为 OFFLINE。修改文件名称，然后修改该表空间对应的数据文件，最后再将 USERS 的状态设置为 ONLINE，如下面例子所示。

例子 4-67 修改数据文件名。

```
SQL> alter tablespace users offline;

表空间已修改。
```

例子 4-68 修改文件名。

```
SQL> host copy E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF to
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS NEW.DBF;
已复制         1 个文件。
```

例子 4-69 修改 Primary 数据库端数据字典中的文件路径。

```
SQL> alter tablespace users rename datafile
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF' to
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS NEW.DBF';
```

```
表空间已修改。
```

例子 4-70 查询数据文件重命名结果。

```
SQL> select name
  2  from v$datafile;

NAME
--------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS_NEW.DBF
```

从输出可以看出在 Primary 数据库端已经成功重命名数据文件。

2. 在 Standby 数据库端

需要先停止 Redo 应用，关闭数据库修改数据文件名，然后启动 Standby 数据库到 MOUNT 状态，修改数据字典记录的文件名再重新启动 Redo 应用。整个步骤通过下面例子演示。

例子 4-71 停止 Standby 数据库端的 Redo 应用。

```
SQL> alter database recover managed standby database cancel;

数据库已更改。

SQL>shutdown
```

例子 4-72 修改文件名。

```
SQL> host copy E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF to
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS NEW.DBF;
已复制         1 个文件。
```

例子 4-73 启动数据库到 MOUNT 状态。

```
SQL> startup mount;
ORACLE 例程已经启动。
......
数据库装载完毕。
```

例子 4-74 修改数据文件在数据字典中的记录。

```
SQL> alter tablespace users rename datafile
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS01.DBF' to
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS_NEW.DBF';

表空间已修改。
```

例子 4-75 重启 Redo 应用。

```
SQL> alter database recover managed standby database disconnect from session;

数据库已更改。
```

为了验证重命名结果，我们在 Standby 数据库端查询数据文件名，如下例所示。

例子 4-76　查询 Standby 数据库端数据文件名。

```
SQL> select name
  2  from v$datafile;

NAME
--------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSTEM01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\USERS_NEW.DBF
```

输出说明，在 Standby 数据库端我们已经重命名了数据文件 USERS 为 USERS_NEW。

4.8.7　添加或删除重做日志组

在物理 Standby 数据库上添加或删除重做日志组是一个独立的行为，与 Primary 数据库无关，只需要在创建或删除之前先停止 Redo 应用，参数 STANDBY_FILE_MANAGEMENT 需要修改为 MANUAL。这里就不给出具体事例，只给出步骤和相关说明。

01 停止 Redo 应用。

02 修改参数 STANDBY_FILE_MANAGEMENT 为 MANUAL。

```
alter system set standby_file_management=manual
```

03 增加或删除日志组，相应的操作采用如下语法进行。

增加日志组：

```
alter database add standby logfile group group_number file
```

删除日志组：

```
alter database drop standby logfile group group_number
```

04 恢复参数 STANDBY_FILE_MANAGEMENT 的初始值。

```
alter system set standby_file_management=auto
```

05 重启 Redo 应用。如果此时创建了 Standby 重做日志，而使用 Redo 的实时应用，可以使用如下方式启动 Redo 应用。

```
SQL> alter database recover managed standby database using current logfile
disconnect from session;

数据库已更改。
```

4.8.8　监控 Data Guard 数据库视图

数据字典视图一直是 Oracle 提供的强有力的管理和监控数据库的工具，在管理 Data Guard 方面，Oracle 提供了大量的管理和监控 Primary 数据库和 Standby 数据库的数据字典。下面我们先给

出一个列表，最后再给出几个重要的数据字典视图的应用演示。

如表 4-2 所示，最左边的 Primary 数据库行为设计一系列的数据库事件，而这些事件发生时，在 Primary 数据库和 Standby 数据库端设计如何查看这些事件引发的变化，可以通过“Primary 数据库数据信息路径”和“Standby 数据库信息路径”涉及的数据字典或文件获得。

表 4-2 数据库事件对应的数据字典和文件

Primary 数据库行为	Primary 数据库数据信息路径	Standby 数据库信息路径
启动和禁止 Redo	Alert.log V$THREAD	Alert.log
关于数据库角色，保护模式，角色切换等信息	V$DATABASE	V$DATABASE
删除或增加日志组	Alert.log V$LOG V$LOGFILE	Alert.log
创建控制文件	Alert.log	Alert.log
监控重做日志	Alert.log V$ARCHIVE_DEST_STATUS	Alert.log V$ARCHIVED_LOG V$LOG_HISTORY V$MANAGED_STANDBY
表空间状态变化	V$RECOVER_FILE DBA_TABLESPACE Alert.log	V$RECOVER_FILE DBA_TABLESPACE
删除或新增数据文件或表空间	DBA_DATA_FILES Alert.log	V$DATAFILE Alert.log
重命名数据文件	V$DATAFILE Alert.log	V$DATAFILE Alert.log
未被日志记录或未回复操作	V$DATAFILE V$DATABASE	Alert.log
监视 Redo 传输	V$ARCHIVE_DEST_STATUS V$ARCHIVE_LOG V$ARCHIVE_DEST Alert.log	V$ARCHIVED_LOG Alert.log
修改初始化参数	Alert.log	Alert.log
执行 OPEN RESETLOGS 或者 CLEAR UNARCHIVED LOGIFILE 语句	Alert.log	Alert.log

1. V$DATABASE

该视图可以查询数据库保护模式、保护级别、数据库角色，以及 SWITCHOVER 状态等信息，如下例所示。

例子 4-77 查询当前数据库的保护模式和数据库角色等信息。

```
    SQL>   select   protection mode,protection level,database role,switchover
status
```

```
  2  from v$database;

PROTECTION_MODE PROTECTION_LEVEL  DATABASE_ROLE SWITCHOVER_STATUS
----------------- ----------------- ------------- -----------------
MAXIMUM PERFORMANCE  MAXIMUM PERFORMANCE  PRIMARY  SESSIONS ACTIVE
```

当前数据库角色为 pirmary，处于最大性能保护模式。当前的 SWITCHOVER 状态为 SESSION ACTIVE，该状态是正常状态说明该数据库可以切换为物理 Standby。

2. V$MANAGED_STANDBY

该视图只对 Standby 数据库有效，可以查询当前的相关数据库进程信息，如进程状态、涉及的日志序列号、数据块等，如下例所示。

例子 4-78 查询 Standby 数据库的进程信息。

```
SQL> select process,client_process,sequence#,status
  2  from v$managed_standby;

PROCESS      CLIENT_P              SEQUENCE#    STATUS
------- ---------- ---------- ------------ ---------- ----------
ARCH         ARCH                  123          CLOSING
ARCH         ARCH                  122          CLOSING
RFS          LWGR                  124          IDLE
RFS          N/A                     0          IDLE
```

其中 PROCESS 表示进程名称，CLIENT_PROCESS 显示和 PROCESS 对应的数据库主进程，如 RFS 进程使用 LWGR 主进程把 Redo 数据写入到 Standby 数据库。

3. V$ARCHIVED_LOG

该视图显示逻辑 Standby 数据库已经归档的日志信息，如下例所示。

例子 4-79 查询逻辑 Standby 已归档日志文件信息。

```
SQL>select name sequence#,applied,completion time
  2  from v$archived_log;
NAME                                          SEQUENCE# APP  COMPLETION TIM
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\LOG1_122_652301232.ARC   122 YES  20-5月 -10
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\LOG1_123_652301232.ARC   123 YES  20-5月 -10
```

4. V$DATAGUARD_STATUS

该视图记录了一系列 Data Guard 事件信息，这些信息会同时记录到 Alert 日志文件中，如下例所示。

例子 4-80 查询当前 Data Guard 记录的事件信息。

```
SQL> select message
  2  from v$dataguard_status;

MESSAGE
----------------------------------------
ARC0: Archival started
ARC1: Archival started
```

```
ARC1: Becoming the 'no FAL' ARCH
ARC1: Becoming the 'no SRL' ARCH
ARC2: Archival started
ARC0: Becoming the heartbeat ARCH
ARCH shutting down
ARC2: Archival stopped

已选择 8 行。
```

从输出可以看出，两个归档进程 ARC2 和 ARC1 分别被启动以及 ARC2 进程被关闭的信息。

4.8.9 设置 Data Guard 保护模式

Data Guard 支持三种数据保护模式即最高可用性（Maximum Availability）、最大性能（Maximum Performance）和最大保护（Maximum Protection）。默认的 Data Guard 数据库保护模式为最大性能。在需要的时候可以修改数据保护模式，注意这里的操作都是在 Primary 数据库上执行。

下面给出设置 Data Guard 数据保护模式的步骤和设置过程中的注意事项。

01 选择保护模式 Maximum Availability。

02 确认 log_archive_dest_n 参数包含三个 Redo 传输服务参数，AFFIRM、SYNC、DB_UNIQUE_NAME。

我们在配置物理 Standby 数据库时，已经在初始化参数中设置了参数 LOG_ARCHIVE_DEST_2，该参数必须包含要设置的数据保护模式的三个属性，如表 4-3 所示。

表 4-3 不同数据保护模式的 Redo 传输属性

最大可用性	最大性能	最大保护
AFFIRM	NOAFFIRM	AFFIRM
SYNC	ASYNC	SYNC
DB_UNIQUE_NAME	DB_UNIQUE_NAME	DB_UNIQUE_NAME

通过上表可以发现需要同步 SYNC 的保护模式都需要确认 AFFIRM，显然这也是同步自身的特性，就像 TCP 可靠传输一样，通过确认实现同步。下面我们给出一个例子，说明如何在一个已经配置好保护模式的 Primary 数据库上修改它的数据保护模式。

在修改之前，我们要确认 Data Guard 的环境配置，如 log_archive_dest_n 参数设置、db_unique_name 设置等，如下例所示。

例子 4-81 查询 log_archive_dest 参数。

```
SQL> show parameter log archive dest 2;

NAME                         TYPE        VALUE
---------------------------- ----------- ----------------------------
log archive dest 2           string      service=lszpri lgwr async
                                         valid for=(online logfiles,pri
                                         mary_role) db_unique_name=lszpri
```

在上述参数 log_archive_dest_2 中使用了 async 异步属性，显然这是最大性能保护模式的属性要求。

03 确认 DB_UNIQUE_NAME 的唯一性，如下例所示。

例子 4-82 查询 Standby 数据库端 db_unique_name 值。

```
SQL> show parameter db unique name;

NAME                          TYPE          VALUE
---------------------- ----------- ----------
db_unique_name                string        lszstdby
```

例子 4-83 查询 Primary 数据库端 db_unique_name 值。

```
SQL> show parameter db unique name;

NAME                                  TYPE         VALUE
------------------------------------ ----------- --------
db_unique_name                       string       lszpri
```

从上两例查询知道两个数据库的 DB_UNIQUE_NAME 具有唯一性。

04 确认 LOG_ARCHIVE_CONFIG 包含 Data Guard 中的所有 db_unique_name，如下例所示。

例子 4-84 查询 LOG_ARCHIVE_CONFIG 参数配置。

```
SQL> show parameter log_archive_config;

NAME                          TYPE        VALUE
---------------------- --------- ----------------------------
log_archive_config            string      dg_config=(lszpri,lszstdby)
```

05 关闭数据库启动到 MOUNT 状态，如下例所示。

例子 4-85 启动数据库到 MOUNT 状态。

```
SQL> shutdown immediate
…
SQL> startup mount;
ORACLE 例程已经启动。
......
数据库装载完毕。
```

06 设置保护模式，如下例所示。

例子 4-86 修改 Primary 数据库的保护模式。

```
SQL> alter database
  2  set standby database to maximize availability;

数据库已更改。
```

此时，我们将当前的 Primary 数据库的数据保护模式修改为最高可用性模式。

07 打开数据库并验证修改结果，如下例所示。

例子 4-87 打开数据库并验证数据保护模式修改结果。

```
SQL> alter database open;

数据库已更改。
SQL> select protection_mode,database_role
  2  from v$database;

PROTECTION_MODE      DATABASE_ROLE
-------------------- --------------------
MAXIMUM AVAILABILITY PRIMARY
```

显然，从输出看出当前的 Primary 数据库的数据保护模式 PROTECTION_MODE 为最高可用性保护模式，即 MAXIMUM AVAILABILITY。

4.9 创建逻辑Standby数据库

创建逻辑 Standby 数据库比创建物理 Standby 数据库要复杂一些，复杂性体现在逻辑 Standby 必须考虑一系列的限制条件。本节我们先介绍 SQL 应用的局限以及如何在逻辑 Standby 端保证唯一标识被更新的数据行，然后通过一个实例演示如何创建一个逻辑 Standby 数据库。

4.9.1 理解 SQL 应用的局限

在创建逻辑 Standby 数据库前，我们先统一认识对逻辑 Standby 的诸多限制。在创建逻辑 Standby 前，同样需要在 Primary 数据库端启动归档、启动 forced logging、创建逻辑 Standby 控制文件等操作。关键是逻辑 Standby 与物理 Standby 的本质区别是逻辑 Standby 使用 SQL 应用实现与 Primary 数据库的同步，这样对 SQL 的操作以及操作的数据类型就有了诸多限制。下面我们分别介绍逻辑 Standby 不支持的行为。

1．逻辑 Standby 不支持的数据类型

```
BFILE
Collections
多媒体数据类型。
ROWID UROWID
用户定义类型
存储关系型对象的 XML 类型
二进制 XML
嵌套表
VARRAYS。
```

2．逻辑 Standby 不支持的表类型

```
启动段压缩的表。
包含存储安全文件的 LOG 列的表。
具有虚列的表。
```

3．逻辑 Standby 不支持的 pl/sql 包

对于修改系统元数据的 PL/SQL 包 SQL 应用都不支持，如下例所示。

```
DBMS_JAVA
DBMA_REGISTRY
DBMS_ALTER
DBMS_SPACE_ADMIN
DBMS_REFRESH
DBMS_REDEFINITION
DBMS_AQ
DBMS_SCHEDULER 等。
```

但是需要注意，在 Oracle 11g 中上述 PL/SQL 包已获得支持，如 DBMS_SCHEDULER 包已经允许在 Standby 数据库上运行 job。读者可以参考 Oracle 的相关白皮书。

4．逻辑 Standby 不支持的 SQL 语句

```
ALTER DATABASE
ALTER MATERIALIZED VIEW
ALTER MATERIALIZED VIEW LOG
ALTER SESSION
ALTER SYSTEM
ALTER CONTROL FILE
CREATE DATABASE
CREATE DATABASE LINK
CREATE PFILE FROM SPFILE
CREATE MATERIALIZED VIEW
CREATE MATERIALIZED VIEW LOG
CREATE SCHEMA AUTHORIZATION
CREATE SPFILE FROM PFILE
DROP DATABASE LINK
DROP MATEARIALIZED VIEW
DROP MATEARIALIZED VIEW LOG
EXPLAIN
LOCK TABLE
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
```

这些 SQL 语句会被 SQL 应用自动跳过而不被执行。

从上面的分析我们可以看出，使用逻辑 Standby 确实存在诸多“不便”，要求对数据类型、SQL 操作类型以及 pl/sql 包类型等提出了限制条件。在读者创建自己的逻辑 Standby 时，必须提前对自己的数据库结构、表数据库类型以及 pl/sql 包的使用等有清晰的认识，否则逻辑 Standby 数据库就无法起到数据保护的作用。

4.9.2　如何唯一标识逻辑 Standby 中的表行

逻辑 Standby 数据库的物理组织结构与 Primary 数据库的物理组织结构不同，有时我们是通过 Primary 数据库的备份来创建逻辑 Standby，但是二者的物理组织结构仍不相同。在逻辑 Standby 端执行诸如 insert、update 等语句时，往往需要唯一定位相关的数据行，此时就需要 Primary 数据库有一种机制来保证这个 SQL 语句可以唯一标识一个数据行。注意此时的 ROWID 不能用于唯一标识数据行，因为逻辑 Standby 无法映射这种 ROWID 到具体的数据块。所以需要一种机制使得 Primary

数据库端可以唯一识别要更新的数据行。

Oracle 使用主键、唯一约束和索引补充日志来逻辑地识别在逻辑 Standby 数据库端的数据行。如果启动了数据库级别的主键、唯一约束和索引补充日志，每一个 UPDATE 语句会连同列值一起来唯一标识被修改的数据行。具体规则如下。

- 如果表具有主键，则主键和被修改列一起作为 UPDATE 语句一部分来辨识别唯一的被修改数据行。
- 如果没有主键，则最短的非空唯一约束或索引和被修改列一起作为 UPDATE 语句一部分来识别唯一的被修改数据行。
- 如果没有主键，也没有非空唯一约束或索引所有可定长度的列和被修改列一起作为 UPDATE 语句一部分来识别唯一的被修改数据行。

基于以上原因，Oracle 推荐对表增加主键或非空唯一索引，总之要保证 SQL 应用 Redo 数据来更新逻辑 Standby 数据库。

可以通过如下语句，在 Primary 数据库端找出没有唯一标识符的表，这些表不被 SQL 支持。

```
SQL>select owner,table name
2 > From dba logstdby not unique
3> Where (owner,table name) not in
4> (select distinct owner,table name from dba logstdby unsupported)
5>  And bad_colum='Y'
```

4.9.3 创建逻辑 Standby 数据库

本节我们讲解创建物理 Standby 的数据库的具体步骤，我们采取的步骤是在 Standby 数据库的基础上创建逻辑 Standby。这里需要读者注意的是参数 log_archive_dest_n 的设置，因为逻辑 Standby 有自己的 Standby 重做日志文件，其归档需要单独的目录，而且 log_archive_dest_1 的参数中 valid_for 属性也需要修改，因为该参数的 location 值指定的目录仅用于存储本地重做日志的归档。

1. 创建物理 Standby 数据库

首先创建一个物理 Standby 数据库，在物理 Standby 的基础上创建逻辑 Standby 数据库。我们使用在 4.5.2 节创建的物理数据库 lszstdby 作为创建逻辑 Standby 的基础，即当前的 Data Guard 环境中有一个 Primary 数据库，一个物理 Standby 数据库。

2. 关闭 Standby 数据库的 Redo 应用

在把物理 Standby 数据库转换为逻辑 Standby 数据库之前，必须先关闭 Redo 服务，即停止物理 Standby 的 Redo 应用。这样可以防止应用保护 LogMiner 字典的 Redo 数据，如果不停止 Redo 应用，在转换为逻辑 Standby 数据库后，使得 SQL 应用时由于 Redo 数据中缺少 LogMiner 字典元数据而影响逻辑 Standby 数据库与 Primary 数据库的数据同步。

在物理 Standby 数据库上停止 Redo 应用，如下例所示。

例子 4-88 停止 Redo 应用。

```
SQL> alter database recover managed standby database cancel;
```

```
数据库已更改。
```

3. 设置 Primary 数据库初始化参数

在创建物理 Standby 数据库时，我们设置了 Primary 数据库端的初始化参数，使得 Primary 数据库的 Redo 数据可以传输到 Standby 数据库。

在创建逻辑 Standby 数据库时，如果希望 Primary 数据库可以进行角色转换，比如可以将 Primary 数据库转换为逻辑 Standby 数据库，此时修改需要 log_archive_dest_1 参数的 valid_for 属性，并添加 log_archive_dest_3 参数。如下代码所示。

```
#参数 log_archive_dest_1 用于将本地重做日志数据写入本地目录存储
log_archive_dest_1='location=e:\oracle\product\10.2.0\oradata\lszpri
valid_for=(online_logfiles,all_roles) db_unique_name=lszpri'
log_archive_dest_state_1=enable

#参数 log_archive_dest_2 用于将本地重做日志数据异步传送到远端 Standby 数据库
log_archive_dest_2='service=lszstdby lgwr async
valid_for=(online_logfiles,primary_role) db_unique_name=lszstdby'
log_archive_dest_state_2=enable

#参数 log_archive_dest_3 用于角色切换
log_archive_dest_3='location=e:\oracle\product\10.2.0\oradata\lszpri_1
valid_for=(standby_logfiles,standby_role) db_unique_name=lszpri'
log_archive_dest_state_3=enable
```

上述代码中的粗体内容就是我们在 Primary 数据库端对初始化参数所做的修改，其中参数 log_archive_dest_3 为新添加的参数，上述的参数设置使得 Primary 数据库可以转换为逻辑 Standby 数据库。

我们可以通过 ALTER SYSTEM SET 指令来修改或添加参数，在 Primary 数据库端执行如下操作。

例子 4-89 修改参数 log_archive_dest_1。

```
SQL>  alter  system  set  log_archive_dest_1='location=e:\oracle\product\
10.2.0\oradata\lszpri
  2  valid_for=(online_logfiles,all_roles) db_unique_name=lszpri'
3  scope=both;

系统已更改。
```

例子 4-90 修改参数 log_archive_dest_3。

```
SQL> alter system set log_archive_dest_3='location=e:\oracle\
    product\10.2.0\oradata\lszpri_1
  2  valid_for=(standby_logfiles,standby_role) db_unique_name=lszpri'
  3  scope=both;

系统已更改。
```

上面例子中，Primary 数据库端日志归档参数的含义，如表 4-4 所示。

表 4-4 Primary 数据库端日志归档参数含义

日志归档参数	当前数据库为 Primary 角色	数据库为逻辑 Standby 角色
log_archive_dest_1	将 Primary 数据库的本地重做日志数据归档存入如下目录。e:\oracle\product\10.2.0\oradata\lszpri	将逻辑 Standby 数据的本地在线重做日志数据归档存入如下目录。e:\oracle\product\10.2.0\oradata\lszpri
log_archive_dest_2	将 Redo 数据通过网络传输到逻辑 Standby 数据库 lststdby。	该参数被忽略
log_archive_dest_3	该参数被忽略	将逻辑 Standby 数据库从 Primary 数据库接收到的重做日志数据归档存入如下目录。e:\oracle\product\10.2.0\oradata\lszpri_1



如果不希望将 Primary 数据库转化为逻辑 Standby 数据库，则不需要做任何参数修改，保持创建物理 Standby 数据库时的 Primary 数据库初始化参数即可。

接着需要在 Primary 数据库端生成 LogMiner 字典信息。在逻辑 Standby 环境下，LogMiner 字典信息必须写入 Redo 数据，这样 SQL 应用的相应 LogMiner 组件才可以正确解释 Redo 数据。在创建 LogMiner 字典的过程中，会自动建立补充日志并记录主键和唯一索引列。补充日志可以唯一包含足够的信息用于辨别被修改的数据行。

因为如果 Redo 数据在 Standby 端为 SQL 应用，如果不能唯一标识数据应用对应的行，则会造成数据同步失败。下面演示如何创建 LogMiner 字典。

例子 4-91 创建 LogMiner 字典。

```
SQL> execute dbms_logstdby.build;

PL/SQL 过程已成功完成。
```

4．转换物理 Standby 为逻辑 Standby

在物理 Standby 数据库上执行操作将物理 Standby 为逻辑 Standby。为了保险期间，我们先看看物理 Standby 数据库的参数 db_name 的值，因为转换到逻辑 Standby 时，需要指定一个全局唯一的数据库名，不能与 Primary 数据库的 db_name 相同，如下例所示。

例子 4-92 查询 Primary 数据库的 db_name。

```
SQL> show parameter db name;

NAME                     TYPE        VALUE
------------------ ---------- -----------
db_name                  string      lszpri
```

从输出看出当前的 db_name 为 lszpri，所以在转换到逻辑数据库时，需要设置一个不同的数据库名，如下例所示。

例子 4-93　转换为逻辑 Standby 数据库。

```
SQL> alter database recover to logical standby lszstdby;

数据库已更改。
```

上述指令中的最后一个参数就是新的逻辑 Standby 数据库的 db_name，因为逻辑 Standby 是一个全新的数据库，它与 Primary 数据库一样提供相应的数据库服务。

下面我们重启数据库，并启动到 MOUNT 状态，如下例所示。

例子 4-94　重启数据库。

```
SQL> shutdown immediate;
ORA-01507: 未装载数据库


ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                 1250380 bytes
Variable Size            163580852 bytes
Database Buffers         432013312 bytes
Redo Buffers               7135232 bytes
数据库装载完毕。
```

接着，我们查看修改后的 db_name，如下例所示。

例子 4-95　查看修改后的逻辑 Standby 的数据库名。

```
SQL> show parameter db name;

NAME                TYPE      VALUE
--------------- -------- -----------
db_name             string    LSZSTDBY
```

从输出看出，此时的 db_name 已经修改成功。下面我们再查看一下此时的数据库角色，如下例所示。

例子 4-96　查看当前数据库的角色。

```
SQL> select database_role
  2  from v$database;

DATABASE_ROLE
-----------------------------
LOGICAL STANDBY
```

在 Standby 数据库端还需要重新创建密码文件。此时的密码必须和 Primary 数据库的 SYS 用户密码相同，因为 Redo 传输依然需要修改密码进行网络通信。创建密码如下例所示。

例子 4-97 创建密码文件。

```
C:\>orapwd      file=e:\oracle\product\10.2.0\db_1\database\PWDlszstdby.ora
password=oracle entries=30
```

在 Oracle 11g 版本中不需要改步骤，按照白皮书的说法是创建新的逻辑 Standby 的密码文件会引起 Redo 传输服务的异常。

5. 设置逻辑 Standby 的初始化参数

因为逻辑 Standby 数据库是一个对外提供服务的数据库，有自己的重做日志和 Standby 重做日志，初始化参数中的 log_archive_dest_1 的 valid_for 属性需要修改，使得 location 指定的目录仅用于存储本地重做日志产生的归档文件，即将 valid_for 属性中的 all_logfiles 值改为 online_logfiles。修改后的部分如下代码所示。

```
log archive dest 1='location= E:\oracle\product\10.2.0\oradata\lszstdby
valid for=(online logfiles,all roles) db unique name=lszstdby'
log archive dest state 1=enable

#如下添加内容用于切换到 primary 角色。
log archive dest 2='service=lszpri lgwr async
valid for=(online logfiles,primary role) db unique name=lszpri'
log archive dest state 2=enable

log archive dest 3='location=e:\oracle\product\10.2.0\oradata\lszstdby 1
valid for=(standby logfiles,standby role) db unique name=lszstdby'
log_archive_dest_state_3=enable
```

在上述代码中，注意参数 log_archive_dest_3 的 valid_for，其参数含义是如果当前的数据库（参数文件对应的数据库）为逻辑 Standby 数据库，则 location 参数的目录用于存储 Standby 重做日志，在逻辑 Standby 数据库上要求创建 Standby 重做日志，这个日志中的信息是从 Primary 数据库通过 SQL 应用得来的。下面我们修改和设置逻辑 Standby 端的初始化参数。

例子 4-98 修改 Standby 数据库端的 log_archive_dest_1 参数。

```
SQL> alter system set log_archive_dest_1='location=e:\
    oracle\product\10.2.0\oradata\lszstdby\
  2   valid_for=(online_logfiles,all_roles) db_unique_name=lszstdby';

系统已更改。
```

例子 4-99 设置 Standby 数据库端的 log_archive_dest_3 参数。

```
SQL> alter system set log_archive_dest_3='location=e:\
    oracle\product\10.2.0\oradata\lszstdby_1
  2   valid_for=(standby_logfiles,standby_roles) db_unique_name=lszstdby';

系统已更改。
```

6. 创建逻辑 Standby 数据库的 Standby 重做日志

逻辑 Standby 数据库需要创建 Standby 重做日志，该日志文件用来接收从 Primary 数据库传输来的 Redo 数据。我们创建三个 Standby 重做日志组，如下例所示。

例子 4-100　创建 Standby 重做日志组。

```
SQL> alter database add standby logfile group 4
('E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY01.LOG')
  2  size 50m;

数据库已更改。
SQL> alter database add standby logfile group 5
('E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY02.LOG')
  2  size 50m;

数据库已更改。
SQL> alter database add standby logfile group 6
('E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY03.LOG')
  2  size 50m;

数据库已更改。
```

例子 4-101　查询当前的逻辑 Standby 数据库端的重做日志信息。

```
SQL>  select group#,member,status
  2   from v$logfile
  3  order by group#;

    GROUP# MEMBER                                              STATUS
---------- ---------------------------------------- ----------------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\REDO01.LOG
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\REDO02.LOG
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\REDO03.LOG
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY01.LOG
         5 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY02.LOG
         6 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZSTDBY\STANDBY03.LOG

已选择 6 行。
```

前三个重做日志组用于记录用户的本地操作，而后三个重做日志组为 Standby 重做日志组，用于存储从 Primary 数据库传输的 Redo 数据。

7．打开逻辑数据库并启动 Redo 服务

例子 4-102　启动数据库。

```
SQL> alter database open resetlogs;

数据库已更改。
```

例子 4-103　启动 Redo 服务。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

此时，我们使用 v$logstdby_state 数据字典视图查看当前是否启动 SQL 应用，如下例所示。

例子 4-104 查看当前是否启动 SQL 应用。

```
SQL> select *
  2  from v$logstdby_state;

PRIMARY_DBID SESSION_ID REALTIME_APPLY                   STATE
-------------- ---------------------------------- -------------------
 2403191133          1      Y                            APPLYING
```

第一个参数 PRIMARY_DBID 是新的数据库 ID，我们知道逻辑 Standby 数据库是新的数据库，所以具有不同于 Primary 数据库的 DBID。这里第三个参数 REALTIME_APPLY 的含义是启动了 SQL 的实时应用，即一旦 standby redo logfile 接收到 Primary 数据库传输过来的 Redo 数据就写入 Standby 数据库。而不用等待 Primary 端的整个重做日志数据全部传输到 Standby 后，再实现 SQL 应用而将 Redo 数据写入 Standby 数据库。第四个参数 STATE 为 APPLYING 说明启用了 SQL 应用。

8．测试逻辑数据库的 Redo 服务

在 Primary 数据库上向 SCOTT 用户的 DEPT 表中插入一行新数据，我们先查看看表中的数据。

例子 4-105 查看表 DEPT 中的数据。

```
SQL> select *
  2  from scott.dept;

   DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

这里的查询是为了验证我们下面的插入数据，当前数据只有 4 行记录，最大的 DEPTNO 为 40。

例子 4-106 向表 SCOTT.DEPT 插入一行记录。

```
SQL> insert into scott.dept
  2  values(50,'MARKETING','BEIJING');

已创建 1 行。
SQL> commit;

提交完成。
```

例子 4-107 强制日志切换。

```
SQL> alter system switch logfile;

系统已更改。
```

因为是异步传输 Redo 数据，我们强制切换日志触发 Redo 传输，而在 Standby 数据库端会实时应用该数据到 Standby 数据库。

在逻辑 Standby 数据库端查询表 DEPT 的变化。

例子 4-108 在 Standby 数据库端查看表 DEPT 中的数据。

```
SQL> select *
  2  from scott.dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        50 MARKETING      BEIJING
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

显然，在 Standby 数据库端，已经有了数据更新，即在 Primary 数据库端的 Redo 数据已经应用到了 Standby 数据库，当前多了一行我们在 Primary 数据端插入的记录。

4.10　逻辑Standby的角色转换

本节我们讲解如何在逻辑 Standby 数据库上实现 Switchover 和 Failover。在切换之前一定要验证切换条件是否满足，我们通过具体的实例演示切换过程。

4.10.1　逻辑 Standby 的 Switchover

1. 验证切换条件是否具备

其实，我们在创建逻辑 Standby 时已经考虑了 Switchover 转换的问题，设置了参数 fal_server、fal_client 以及 log_archive_dest_n。下面我们检查 Primary 数据库的这些参数设置。

例子 4-109 检查 Primary 数据库 fal_server、fal_client 参数。

```
SQL> show parameter fal

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
fal_client                           string      lszpri
fal_server                           string      lszstdby
```

因为当前的数据库为 lszpri，所以转换后 Primary 数据库 lszpri 就成为逻辑 Standby 数据库，所以此时的 fal_server 参数为 lszstdby。

检查 db_file_name_conver 和 log_file_name_convert 参数，如下例所示。

例子 4-110 检查文件名转换参数。

```
SQL> show parameter file_name_convert;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_name_convert                 string      oradata\lszstdby, oradata\lszpri
log_file_name_convert                string      oradata\lszstdby, oradata\lszpri
```

接着，我们查看当前 Primary 数据库的 log_archive_dest_n 参数的设置是否满足切换条件。

例子 4-111 查看所有 log_archive_dest_n 参数。

```
SQL> show parameter log_archive_dest

NAME                     TYPE       VALUE
---------------------    ---------- -----------------------------
log_archive_dest         string
log_archive_dest_1       string     location=e:\oracle\product\10.
                                    2.0\oradata\lszpri
                                    valid_for=(online_logfiles,al
                                    l_roles) db_unique_name=lszpri
log_archive_dest_10      string
log_archive_dest_2       string     service=lszstdby lgwr async
                                    valid_for=(online_logfiles,pri
                                    mary_role) db_unique_name=lszstdby
log_archive_dest_3       string     location=e:\oracle\product\10.

NAME                     TYPE       VALUE
------------------------ ---------- -----------------------------
                                    2.0\oradata\lszpri_1
                                    valid_for=(standby_logfiles,s
                                    tandby_role) db_unique_name=lszpri
```

这里关键是 log_archive_dest_3 参数，它说明当前的 Primary 数据库转换为 Standby 角色后，作为 Standby 角色，它接收到的远端重做日志的数据要存放在 location 属性指定的目录下。在转换为 Standby 角色后，参数 log_archive_dest_2 被忽略。如果读者在创建逻辑 Standby 数据库时，没有考虑 Primary 数据库的角色切换问题，就需要使用 alter system set 指令设置参数 log_archive_dest_3，并注意一定要同时设置该参数对应的 log_archive_dest_state_3 为 enable，如下例所示。

例子 4-112 设置参数 log_archive_dest_state_3。

```
    SQL> alter system set log archive dest 3='location=e:\oracle\ product\
10.2.0\oradata\lszpri 1
    valid for=(standby logfiles,standby role) db unique name=lszpri'

    系统已更改
```

对于 Standby 数据库，创建 Standby 重做日志是必须的。所以还需要检查当前的 Primary 数据库是否具有 Standby 重做日志，如下例所示。

例子 4-113 检查数据库是否具有 Standby 重做日志。

```
SQL> select *
  2  from v$standby_log;

未选定行
```

输出显示，在 Primary 数据库上没有创建 Standby 重做日志，下面创建三个 Standby 重做日志，如下例所示。

例子 4-114　增加 Standby 日志组。

```
SQL> alter database add standby logfile group 4
  2  ('e:\oracle\product\10.2.0\oradata\lszpri\stdby01.log')
  3  size 50m;

数据库已更改。
SQL> alter database add standby logfile group 5
  2  ('e:\oracle\product\10.2.0\oradata\lszpri\stdby02.log')
  3  size 50m;

数据库已更改。
SQL> alter database add standby logfile group 6
  2  ('e:\oracle\product\10.2.0\oradata\lszpri\stdby03.log')
  3  size 50m;

数据库已更改。
```

在 Primary 数据库端创建了三个 Standby 重做日志组后，我们通过数据字典 v$standby_log 来检验创建结果，如下例所示。

例子 4-115　查询当前 Standby 日志信息。

```
SQL> select group#,sequence#,used,archived,status
  2  from v$standby_log;

    GROUP#  SEQUENCE#       USED ARC STATUS
---------- ---------- ---------- --- ----------
         4          0        512 YES UNASSIGNED
         5          0        512 YES UNASSIGNED
         6          0        512 YES UNASSIGNED
```

因为是新建的重做日志组，并且当前的 Primary 数据库还没有切换为 Standby 数据库，所以还没有使用，日志组的日志成员中根本没有数据。

下面我们查看 Standby 日志组中的日志成员信息，和非 Data Guard 数据库环境下一样使用 v$logfile 数据字典，如下例所示。

例子 4-116　查看 Standby 日志组中的日志成员。

```
SQL> col member for a55
SQL> set line 120
SQL>  select group#,member,type,status
  2   from v$logfile
  3  order by group#;

    GROUP# MEMBER                                            TYPE     STATUS
---------- ------------------------------------------------ -------- -------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO01.LOG  ONLINE   STALE
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO02.LOG  ONLINE   STALE
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO03.LOG  ONLINE
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\STDBY01.LOG STANDBY
         5 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\STDBY02.LOG STANDBY
         6 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\STDBY03.LOG STANDBY
```

```
已选择 6 行。
```

从输出可以看出，对于创建的 Standby 重做日志其类型为 Standby，而本地重做日志类型为 ONLINE。读者可以注意到，在我们配置 log_archive_dest_n 参数时，属性 valid_for 中的日志文件类型我们使用了 online_logfiles、standby_logfiles 和 all_roles，其中的 online_logfiles 表示读取本地重做日志数据，而 standby_logfiles 表示写入 Standby 数据库端的 Standby 重做日志。

在 Standby 数据库端也做同样的检查，如下例所示。

例子 4-117 在 Standby 数据库端检查 fal_*参数。

```
SQL> show parameter fal

NAME                              TYPE       VALUE
--------------------------  ----------  ------------------
fal client                        string     lszstdby
fal_server                        string     lszpri
```

显然，这里的 fal_client 与 fal_server 的参数值与 Primary 数据库端的正好相反，因为是二者之间进行 Switchover 角色转换，所以这个参数值设置没问题。

接着，我们检查 log_archive_dest 参数，如下例所示。

例子 4-118 检查相关的 log_archive_dest 参数。

```
SQL>  show parameter log_archive_dest

NAME                         TYPE        VALUE
----------------------  ---------    ---------------------------------
log_archive_dest             string
log_archive_dest_1           string      location=e:\oracle\product\10.
                                         2.0\oradata\lszstdby valid_for
                                         =(online_logfiles,all_roles) d
                                         b_unique_name=lszstdby
log_archive_dest_10          string
log_archive_dest_2           string      service=lszpri lgwr async
                                         valid_for=(online_logfiles,pri
                                         mary_role) db_unique_name=lszp
                                         ri
log_archive_dest_3           string      location=e:\oracle\product\10.

NAME                         TYPE        VALUE
-----------------------  -----------  -----------------------------
                                         2.0\oradata\lszstdby_1
                                         valid_for=(standby_logfiles,s
                                         tandby_roles) db_unique_name=l
                                              szstdby
```

因为当前是在 Standby 数据库端，需要将其转换为 Primary 数据库，所以需要注意参数 log_archive_dest_2，即该 Standby 数据库转化为 Primary 角色后，如何同步 Standby 数据库。如果读者在创建逻辑 Standby 数据库时，没有考虑角色转换问题，则需要再次设置该参数，可以通过 alter system set 指令，也可以通过修改参数文件的方式，读者可以自己选择。

2．准备数据库状态转换

在将逻辑 Standby 数据库转换为 Primary 数据库以进行 Switchover 转换时，需要分别在当前要转换的 Primary 数据库端和 Standby 端做准备工作。我们先在 Primary 数据库上执行准备转换为 Standby 的工作，验证 Switchover 状态，然后在 Standby 数据库上执行准备转换为 Primary 数据库的工作。

首先，我们查看 Primary 数据库当前的 Switchover 状态，如下例所示。

例子 4-119　查看 Primary 数据库当前的 Switchover 状态。

```
SQL>  select switchover status,database role
  2   from v$database;

SWITCHOVER STATUS    DATABASE ROLE
-------------------- ----------------
SESSIONS ACTIVE      PRIMARY
```

以上查询显示，当前的数据库角色还是 Primary，而 SWITCHOVER 状态为 SESSIONS ACTIVE，该状态说明当前的 Primary 数据库可以转换为 Standby。如果参数 SWITCHOVER 状态为 SESSIONS ACTIVE 或 TO STANDBY 都说明 Primary 数据库状态正常，允许执行 Switchover 角色转换。

接着，做准备工作，在 Primary 数据库上将 Primary 数据库设置准备状态，具体操作如下例所示。

例子 4-120　将 Primary 数据库转换为 prepare 状态。

```
SQL> alter database prepare to switchover to logical standby;

数据库已更改。
```

上述指令将 Primary 数据库转换为 prepare 状态，我们再次通过数据字典 v$database 查看 Switchover 的状态，如下例所示。

例子 4-121　查看 Switchover 的状态。

```
SQL>  select switchover_status,database_role
  2   from v$database;

SWITCHOVER_STATUS    DATABASE_ROLE
-------------------- ----------------
PREPARING SWITCHOVER PRIMARY
```

此时的 Switchover 状态已经转换为 PREPARING SWITCHOVER，说明我们准备工作转换成功，此时的数据库角色仍然为 Primary。

接着在 Standby 数据库上，同样需要准备转换为 Primary 数据库，我们通过如下指令实现。

例子 4-122　把 Standby 数据库切换为 Primary 数据。

```
SQL> alter database prepare to switchover to primary;

数据库已更改。
```

为了验证准备转换结果，我们在 Standby 数据库查询 Switchover 的状态，如下例所示。

例子 4-123 在 Standby 数据库查询 Switchover 的状态。

```
SQL> select switchover_status,database_role
  2  from v$database;

SWITCHOVER_STATUS    DATABASE_ROLE
-------------------- ----------------
PREPARING SWITCHOVER LOGICAL STANDBY
```

上述输出说明在 Standby 数据库端所做的准备转换工作成功执行，此时，对应的 Primary 数据库的 Switchover 状态也发生了变化，已经变为允许转换为 Standby 数据库了，如下例所示。

例子 4-124 在 Standby 数据库端查看 Switchover 状态。

```
SQL>select switchover_status,database_role
  2*  from v$database

SWITCHOVER_STATUS    DATABASE_ROLE
-------------------- ----------------
TO LOGICAL STANDBY   PRIMARY
```



逻辑 Standby 执行完 prepare 工作后，会生成作为 Primary 数据库必备的 LogMiner 字典信息。在创建逻辑 Standby 时，我们在 Primary 数据库端实现了该步骤，这里 Switchover 切换同样需要在要转换为 Primary 数据库的 Standby 数据库上执行该步骤，不过这里是通过 prepare 指令潜在执行罢了。

3. 将 Primary 数据库转换为 Standby

因为 Primary 数据库端已经做好了转换为 Standby 的准备，所以，现在可以执行转换了，在 Primary 数据库端执行如下转换指令。

例子 4-125 转换 Primary 数据库为 Standby 数据库。

```
SQL> alter database commit to switchover to logical standby;

数据库已更改。
```

在执行完转换指令后，我们在 Standby 数据库上查看 Switchover 的状态是否变化，如下例所示。

例子 4-126 查看 Standby 数据库的 Switchover 的状态。

```
SQL>  select switchover_status,database_role
  2   from v$database;

SWITCHOVER_STATUS    DATABASE_ROLE
-------------------- ----------------
TO PRIMARY           LOGICAL STANDBY
```

此时的 Standby 数据库端显然允许转换为 Primary 数据库，但是当前的身份仍然是 Logical Standby。

现在可以在 Standby 数据库上执行转换为 Primary 数据库的操作了，如下例所示。

例子 4-127 Standby 数据库转换为 Primary 数据库。

```
SQL> alter database commit to switchover to primary;

数据库已更改。
```

此时的 Standby 数据库已经是 Primary 数据库了，我们在上述指令执行后，继续在当前的数据库上执行如下操作。

例子 4-128 查看数据库角色。

```
SQL>  select switchover_status,database_role
  2   from v$database;

SWITCHOVER_STATUS    DATABASE_ROLE
-------------------- ----------------
SESSIONS ACTIVE      PRIMARY
```

从上述输出看出，已经将 Standby 数据库转换为 Primary 数据库了，而原先的 Primary 数据库转换成了 Standby 数据库。所以此时为了在转换后的 Standby 数据库上执行 SQL 应用，需要在新的逻辑 Standby 上执行 SQL 应用，如下例所示。

例子 4-129 启动实时执行 SQL 应用。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

4.10.2　逻辑 Standby 的 Failover

逻辑 Standby 的 Failover 很简单 ，因为毕竟 Primary 数据库出现故障时的切换，此时可以不考虑 Primary 数据库，直接在逻辑 Standby 数据库上执行。在执行之前需要确认以下几个问题。

1. 前期准备工作

（1）需要确定在逻辑 Standby 数据库上是否丢失归档日志

如果 Primary 数据库依然可以使用，可以通过数据字典 v$archived_log 在 Primary 数据库上查询当前已经归档的日志序列号，然后在 Standby 数据库通过数据字典 dba_logstdby_log 查询当前已经应用的最大的日志序列号，看二者是否相同。若不相同则需要从 Primary 数据库拷贝缺失的响应序列号日志文件到 Standby 数据库，并使用 alter database registster logical logfile 指令注册缺失的日志文件。

如果不确定哪个是缺失的日志文件，可以把你认为可能的日志文件从 Primary 数据库拷贝到 Standby 数据库，最后通过注册指令注册这些文件，Oracle 会自动识别哪些是缺失的日志文件，如下例所示。

```
SQL>alter database register logical logfile
 'E:\oracle\product\10.2.0\oradata\lszstdby\log-%t_%s_%r.arc';
```

（2）确定新 Primary 数据库的远端逻辑 Standby 数据库参数

在创建逻辑 Standby 数据库时，如果已经提前做好了 Failover 的准备，则会自动设置该参数，以告诉新的 Primary 数据库需要将 Redo 数据传输到那个 Standby 数据库。如我们已经有一个 Standby 数据库 lszstdby2，则需要在要切换为 Primary 数据库的 Standby 数据库中配置如下参数。

```
log_archive_dest_2='service=lszstdby2 lgwr async
valid_for=(online_logfiles,primary_role) db_unique_name=lszstdby2'
log_archive_dest_state_2=enable
```

如果没有配置，则需要使用 alter system set 设置该参数，注意一定要将该参数的状态设置为 enable，否则不起作用。

2．切换逻辑 Standby 数据库为 Primary 数据库

在要切换为 Primary 数据库的逻辑 Standby 数据库上做如下操作。

```
SQL>alter database activate logical standby database finish apply

数据库已更改
```

一旦执行该语句，它将停止逻辑 Standby 端的 RFS 进程，应用 Standby 重做日志中的 Redo 数据，停止 SQL 应用并激活逻辑 Standby 数据库的 Primary 角色。

子句 finish apply 的作用是将未应用的、当前的 Standby 日志数据在逻辑 Standby 转换为 Primary 角色前就应用到数据库，否则在逻辑 Standby 转换为 Primary 角色前，未应用的、当前的 Standby 日志数据不会写到数据库中去。

3．修复其他逻辑 Standby 数据库

在逻辑 Standby 数据库 Failover 后，以前的逻辑 Standby 数据库不再是新的 Primary 数据库的 Data Guard 环境的一部分，需要做额外的工作将其加入到新的 Primary 数据库中来。

首先创建需要连接到 Standby 数据库 lszstdby2 上的 DBLINK，如下例所示。

例子 4-130 创建到 Standby 数据库 lszstdby2 上的 DBLINK。

```
SQL>create database link tolszpri  connect to sys identified by oracle using
'lszpri';

数据库链接已创建。
```

然后在逻辑 Standby 数据库上启动 SQL 应用，此时会用到链接到 Primary 数据库的 DBLINK，如下例所示。

例子 4-131 逻辑 Standby 数据库上启动 SQL 应用。

```
SQL>alter database start logical standby apply new primary tolszpri;

数据库已更改。
```

此时，可以检查当前的 SQL 应用是否启动。如果没有启动，很可能的原因是在新的 Primary 数据库端还有归档日志没有传输到 Standby 数据库，造成 Standby 数据库端的归档日志不连续，所以只要把相应的归档日志拷贝过来并注册后，Standby 数据库会自动应用拷贝过来的归档日志。

4.11　管理逻辑Standby数据库

相比物理 Standby 数据库而言，逻辑 Standby 的管理就复杂一些。因为在逻辑 Standby 上存在活跃的 SQL 行为，所以其管理就涉及诸多数据库对象的问题，以及 SQL 应用是否支持的数据类型和操作问题。本节我们介绍管理逻辑 Standby 的数据字典视图以及逻辑 Standby 的各种数据库操作。

4.11.1　限制修改逻辑 Standby 数据库的对象

逻辑 Standby 数据库是一个新的数据库，可以对外提供报表查询甚至数据修改的任务。为了更安全的控制用户的行为，尤其是对 Standby 维护的表的行为，Data Guard 允许用户在一定范围内控制对 Standby 维护的表的控制。

SQL 语句 ALTER DATABASE GUARD 可以控制用户在逻辑 Standby 数据库中访问的表。该指令有三个关键字，含义如下例所示。

1．ALL

ALL 关键字阻止 SYS 用户外的所有用户修改逻辑 Standby 数据库上的任何数据，我们通过下例演示一下 ALL 关键字的作用。我们使用 SYSTEM 用户登录逻辑 Standby 数据库，然后创建一个表 test3。我们先启动 ALL 关键字的 GUARD，如下例所示。

```
SQL> alter database guard all;

数据库已更改。
```

接着，我们试图创建一个表 test3，如下例所示。

例子 4-132 创建表 test3。

```
SQL> conn system/oracle
已连接。
SQL> create table test4 as select * from scott.dept;
create table test4 as select * from scott.dept
                                          *
第 1 行出现错误:
ORA-01031: 权限不足
```

显然提示，权限不足无法实现对表 dept 的访问。说明 GUARD 在保护整个 Standby 数据库中的任何数据不被修改。

2．STANDBY

STANDBY 关键字阻止除了 SYS 用户外的所有用户对 SQL 应用维护的表或序列号进行 DML 和 DDL 操作。修改当前逻辑 Standby 用户访问限制，如下例所示。

例子 4-133 修改逻辑 Standby 用户访问限制。

```
SQL> alter database guard standby;

数据库已更改。
```

此时，我们试图删除 scott 用户的 dept 表，看逻辑 Standby 数据库的“反映”如何，如下例所示。

例子 4-134 删除 scott 用户的 dept 表。

```
SQL> drop table scott.dept;
drop table scott.dept
               *
第 1 行出现错误:
ORA-16224: Database Guard 已启用
```

该表是受 SQL 应用保护的所以，启动了 STANDBY 关键字的 GUARD，所以不会被删除掉。

3. NONE

该关键字说明不做任何 Database Guard 的保护。当然逻辑数据库的数据保护可以在当前的会话中临时地关闭和启动，这通过使用 alter session disable/enable guard 语句实现，如下例所示。

例子 4-135 在当前会话关闭和启动数据库保护。

```
SQL> alter session disable guard;

会话已更改。

SQL> alter session enable guard;

会话已更改。
```



上述的数据库保护的启动与关闭需要在 SYS 或 SYSTEM 用户下操作。

4.11.2 管理和监控逻辑 Standby 数据库视图

在 Oracle 数据库中，视图是一个重要的数据库管理工具，它反映了数据库运行的各种运行状态、性能指标以及文件组成等。掌握适量的逻辑 Standby 数据库视图使得我们可以掌握或控制逻辑 Standby 数据库的运行状态。下面我们将依次介绍这些重要的视图，演示其用法，并给出详细的视图功能解释。

1. DBA_LOGSTDBY_EVENTS

该视图记录了在 SQL 应用过程中发生的事件。一旦在执行 SQL 应用过程中发生错误，就可以通过该视图查看具体事件内容。我们通过例子查看当前的逻辑 Standby 数据库中该视图记录的事件以及状态，如下例所示。

例子 4-136 查看逻辑 Standby 数据库记录的事件以及状态。

```
SQL> col status for a40
SQL> col event for a40
SQL> select event time,status,event
  2  from dba logstdby events;
```

```
EVENT_TIME    STATUS                                              EVENT
---------     ------------------------------------------- ---------------
16-5 月 -10    ORA-16111: log mining and apply setting up
17-5 月 -10    ORA-16111: log mining and apply setting up
17-5 月 -10    ORA-16226: DDL skipped due to lack of support   ALTER DATABASE OPEN
17-5 月 -10    ORA-16128: User initiated stop apply successfully
             completed

17-5 月 -10    ORA-16111: log mining and apply setting up
```

以上输出显示当前数据库的 SQL 应用关闭和启动的信息，以及执行或取消的 DDL 语句的情况。



该视图记录的记录个数默认为 100 条，但是可以通过存储过程修改记录数，即执行存储过程 DBMS_LOGSTDBY.APPLY_SET()。

2．DBA_LOGSTDBY_LOG

该视图动态地记录了 SQL 应用过程中关于归档日志的信息，如归档日志文件名、归档的日志序列号、该归档是否已经应用等，如下例所示。

例子 4-137　查询 SQL 应用关于归档日志的信息。

```
SQL>select file_name,sequence#,first_change#,next_change#,thread#,applied
  2* from dba_logstdby_log

FILE_NAME                 SEQUENCE# FIRST_CHANGE# NEXT_CHANGE#   THREAD# APPLIED
-------------------------- ---------------- ---------- ------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LS    21      797854   798640     1   CURRENT
ZPRI_1\ARC00021_0719026666.001

E:\ORACLE\PRODUCT\10.2.0\ORADATA\LS    22     798640    798643     1    NO
ZPRI_1\ARC00022_0719026666.001
```

其中 APPLIED 参数的含义是当前的归档的重做数据是否已经应用到数据库。如果是 YES 表示已经应用，如果是 NO 表示没有应用，但是这个状态不会保持很长时间，除非你的 SQL 应用出现问题。如果此处的 APPLIED 值是 CURRENT 说明当前的事务涉及多个归档日志文件。

3．V$DATAGUARD_STATS

该视图记录了逻辑 Standby 的 Failover 信息，如何时发生 Failover（apply finish time）、当前提交的数据（apply lag）以及可能的数据丢失（transport lag）等。查询逻辑 Standby 的 Failover 信息如下例所示。

例子 4-138　查询逻辑 Standby 的 Failover 信息。

```
SQL> select name,value,unit
  2  from v$dataguard stats;

NAME                  VALUE              UNIT
-------------------- ------------ ------------------------------
```

```
apply finish time                    day(2) to second(1) interval
apply lag              +00 00:06:58 day(2) to second(0) interval
transport lag           +00 00:00:03 day(2) to second(0) interval
```

上面的输出显示了当前的逻辑 Standby 数据库应用了部分的 Redo 数据。

4．V$LOGSTDBY_PROCESS

该视图显然与进程（process）有关，其实该视图的作用就是记录 SQL 应用中各种进程的当前状态信息，可以通过下例查询关于当前 SQL 应用的部分信息。

例子 4-139 查询 SQL 应用中进程的当前状态信息。

```
SQL> select sid,serial#,spid,type,high_scn
  2  from v$logstdby process;

       SID    SERIAL# SPID         TYPE                            HIGH SCN
---------- ---------- ------------ ------------------------------ ----------
       143         17 2380         COORDINATOR                        799230
       142         32 3596         READER                             799230
       140          3 3900         BUILDER                            799214
       139          3 1712         PREPARER                           799213
       137          2 2664         ANALYZER                           799214
       136          2 3144         APPLIER                            799178
       125          3 3256         APPLIER                            799214
       133          3 3880         APPLIER                            799129
       135          3 1604         APPLIER                            799152
       124          3 288          APPLIER                            799156

已选择 10 行。
```

输出中的 SID、SERIAL#、SPID 唯一标识一个进程身份，TYPE 表示 SQL 应用涉及的进程，HIGH_SCN 表示进程当前操作的最大 SCN。因为页面显示关系，我们继续使用下例查询其他关于当前 SQL 应用的信息。

例子 4-140 查询 SQL 应用中进程的当前状态信息。

```
SQL> select type,status
  2  from v$logstdby process;

TYPE                        STATUS
--------------------------- ---------------------------------------------
COORDINATOR                 ORA-16116: 无可用工作
READER                      ORA-16240: 正在等待日志文件 (线程号 1, 序列号 26)
BUILDER                     ORA-16116: 无可用工作
PREPARER                    ORA-16116: 无可用工作
ANALYZER                    ORA-16116: 无可用工作
APPLIER                     ORA-16116: 无可用工作
APPLIER                     ORA-16116: 无可用工作
APPLIER                     ORA-16116: 无可用工作
APPLIER                     ORA-16116: 无可用工作
APPLIER                     ORA-16116: 无可用工作
```

```
已选择 10 行。
```

在该输出中，STATUS 说明当前进程的状态。我们发现其实当前的逻辑 Standby 很轻松，无事可做，只有 READER 进程在等待从 Primary 数据库传送过来的 Redo 数据。

我们进一步分析 SQL 应用的系统架构图，从而分析 Redo 数据在 Standby 数据库端处理的整个步骤。SQL 应用的流程图如图 4-6 所示。

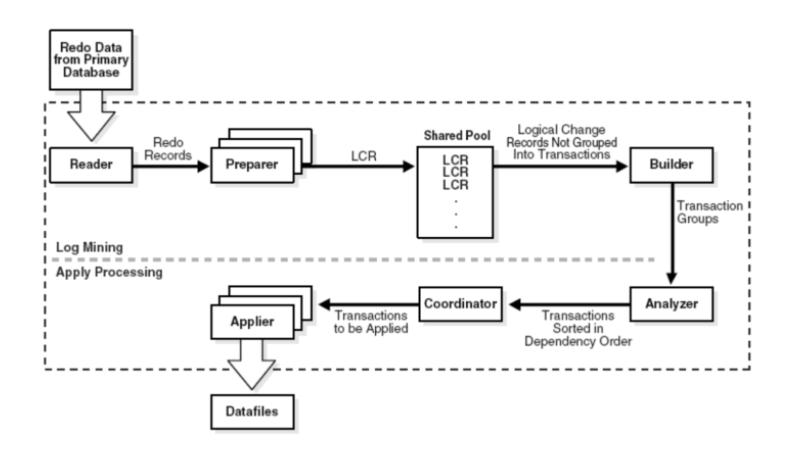


图 4-6　SQL 应用流程图

上图是 SQL 应用的流程，在整个流程中涉及了几个进程，下面我们分别介绍这些进程的作用，以便更好地理解 SQL 应用。

- READER 进程：该进程负责从归档的日志文件或者当前的逻辑 Standby 日志文件读取 Redo 数据。
- PREPARE 进程：该进程做一些“准备工作”，将 Redo 数据转换为逻辑变化号（LCR），对于一个单独的日志文件会有多个 PREPARE 进程工作，而 LCR 信息存储在 SGA 的共享池中。
- BUIDER 进程：该进程把 LCR 组装到对应的事务之中，除此之外该进程还完成诸如 LCR 缓存管理、与 SQL 应用重启相关的检验点事件等任务。
- ANALYZER 进程：该进程识别不同事务之间的依赖关系，将事务依据依赖关系排序。
- COORDINATOR 进程：该进程的作用从字面意思就知道是“协商”进程，它负责协调多个事务之间的依赖关系，并将多个事务分配给不同的 APPLIER 进程，以避免依赖关系造成的相互影响。
- APPLIER 进程：该进程将事务应用到数据库，即最终实现 SQL 应用环节。

我们可以通过数据字典视图 V$LOGSTDBY_PROGRESS 查看当前 SQL 应用进程的状态，也可以通过数据字典视图 V$LOSSTDBY_STATS 查看进程的统计信息。下面我们介绍这些相关的进程。

5．V$LOGSTDBY_PROGRESS

该视图反映了 SQL 应用的当前进程（progress）信息，查询语句如下例所示。

例子 4-141　查询 SQL 应用的当前进程信息。

```
SQL> select *
```

```
  2  from v$logstdby_progress;

APPLIED_SCN APPLIED_TIME  RESTART_SCN RESTART_TIME  LATEST_SCN LATEST_TIME MINING_SCN MINING_TIME
--------- ------------ ------- ---------- ------ --------- ------ -----------
 799229 17-5月 -10   798963 17-5月 -10   800176 17-5月 -10   799230 17-5月 -10
```

以上输出说明已经应用到逻辑 Standby 的 SCN（APPLIED_SCN）为 799299 以及时间为“17-5月 -10”，SQL 应用开始记录的 SCN（RESTART_SCN）以及时间、最后从 Primary 数据库接收到的 SCN（LATEST_SCN）以及时间和 BUILDER 进程处理的最后的记录 SCN（MINING_SCN）。

6. V$LOSSTDBY_STATE

该参数说明当前 SQL 应用的状态，以及是否启动实时 SQL 应用等信息，查询方法如下例所示。

例子 4-142 当前 SQL 应用以及是否启动实时 SQL 应用信息。

```
SQL> col realtime_apply for a20
SQL> col state for a10
SQL> run
  1  select *
  2* from v$logstdby_state

PRIMARY_DBID SESSION_ID REALTIME_APPLY      STATE
---------- ----------------------- ------------------------- ------
 2092838584             1    N                    IDLE
```

从输出可以看出，当前的 DBID 为 2092838584（PRIMARY_DBID），没有启动 SQL 实时应用（REALTIME_APPLY=N），当前状态为 IDLE（STATE）。LogMiner 对应到 SQL 应用的会话 ID 为 1（SESSION_ID）。

如果我们启动在 Primary 数据库端触发一个日志切换，此时的 STATE 状态会发生变化，成为 APPLYING 状态，如下例所示。

例子 4-143 查询 SQL 应用状态。

```
SQL> select *
  2* from v$logstdby state

PRIMARY DBID SESSION ID REALTIME APPLY      STATE
--------- ----------------------- --------------------------- ----------
 2092838584          1 N                   APPLYING
```

注意，APPLYING 状态是暂时的，一旦 SQL 应用完成，该状态自动转换为 IDLE。

视图中列 REALTIME_APPLY 显示当前是否为实时 SQL 应用，我们可以通过如下指令启动当前逻辑 Standby 数据库为实时 SQL 应用，如下例所示。

例子 4-144 启动逻辑 Standby 数据库为实时 SQL 应用。

```
SQL> alter database stop logical standby apply;

数据库已更改。

SQL> alter database start logical standby  apply immediate;
```

```
数据库已更改。
```

此时已经将逻辑 Standby 数据库启动为实时 SQL 应用，即一旦有 Redo 数据写入 Primary 数据库端的重做日志，就会触发 Redo 数据传输到 Standby 数据库端，而不用等待整个事务的 Redo 数据全部写入重做日志再传输到 Standby 数据库端。

我们再次在逻辑 Standby 数据库端查看当前 SQL 应用的状态，如下例所示。

例子 4-145　查看当前 SQL 应用的状态。

```
SQL> select *
  2  from v$logstdby_state;

PRIMARY_DBID SESSION_ID REALTIME_APPLY        STATE
---------- ---------- ---------------------------------------- ----------
 2092838584           1  Y                    IDLE
```

从输出看出视图中列 REALTIME_APPLY 的值已经变为了“Y”，说明已经成功启动了实时 SQL 应用。

7．V$LOGSTDBY_STATS

该视图记录了 SQL 应用的统计信息、当前状态等信息。如果 SQL 应用没有运行，则该视图不会有任何数据显示。显然只有在逻辑 Standby 数据库上，该视图才有意义，因为只有在逻辑 Standby 数据库上才会启动 SQL 应用。

在逻辑 Standby 数据库上查询该视图信息的例子如下所示。

例子 4-146　在逻辑 Standby 数据库上查询视图 v$logstdby_stats。

```
SQL> col name for a40
SQL>select *
  2* from v$logstdby_stats

NAME                                    VALUE
------------------------------------    ------------
number of preparers                     1
number of appliers                      5
maximum SGA for LCR cache               30
parallel servers in use                 9
maximum events recorded                 100
preserve commit order                   TRUE
transaction consistency                 FULL
record skip errors                      Y
record skip DDL                         Y
record applied DDL                      N
record unsupported operations           N

NAME                                    VALUE
------------------------------------    ------------
coordinator state                       IDLE
transactions ready                      30
transactions applied                    30
```

```
coordinator uptime                       756
realtime logmining                       Y
apply delay                              0
Log Miner session ID                     1
txns delivered to client                 306
DML txns delivered                       190
DDL txns delivered                       0
CTAS txns delivered                      0

NAME                                     VALUE
-------------------------------------    ------------
Recursive txns delivered                 116
Rolled back txns seen                    0
LCRs delivered to client                 1106
bytes of redo processed                  1003584
bytes paged out                          0
seconds spent in pageout                 0
bytes checkpointed                       0
seconds spent in checkpoint              0
bytes rolled back                        0
seconds spent in rollback                0
seconds system is idle                   0

已选择 33 行。
```

输出显示了丰富的 SQL 应用的信息，如在 SQL 应用过程中涉及的 APPLIER（number of appliers）和 PREPARE 进程数量（number of preparers）、分配给 LCR 缓存的大小（maximum SGA for LCR cache），以及 DBA_LOGSTDBY_EVENTS 记录的事件数量（maximum events recorded）等。

4.11.3 监控 SQL 应用过程

SQL 应用有六个状态，根据应用环境的不同，SQL 应用状态在六个状态之间转换。其状态转换如图 4-7 所示。

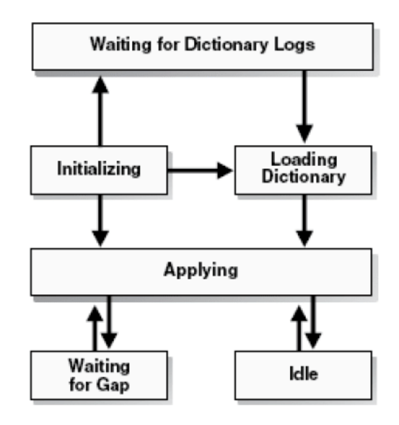


图 4-7 SQL 应用状态转换图

下面我们依次解释这六个状态。

- 初始化状态（Initializing）：当在逻辑 Standby 数据库上启动 SQL 应用时，此时的 SQL 应用就处于 Initializing 状态。这是个短暂的状态。我们也可以通过 V$LOGSTDBY_STATE 视图查询，如下例所示。

例子 4-147　查询 SQL 应用状态。

```
SQL> select session_id,state
  2  from v$logstdby_state;

SESSION_ID STATE
---------- -------------------------
         1 INITIALIZING
```

- 等待字典日志状态（Waiting for Dictionary Logs）：在使用 ALTER DATABASE START LOGICAL STADNBY APPLY 指令启动 SQL 应用时，首先需要装载从 Primary 数据库获得的 LogMiner 字典数据。如果在启动 SQL 应用时，还没有装载 LogMiner 字典，就会处于等待字典日志状态。
- 装载 LogMiner 字典状态（Loading Dictionary）：这个状态说明当前的逻辑 Standby 数据库正在装载 logMiner 字典数据，如果对于大型的 Primary 数据库，逻辑上 Standby 数据库端的过程需要相对较长的一段时间。
- 应用状态（Applying）：该状态说明 SQL 应用已经启动完毕，正在应用 Redo 数据到逻辑 Standby 数据库。查询 v$logstdby_state 其输出如下例所示。

例子 4-148　查询 SQL 应用状态。

```
SQL>  select session_id,state
  2   from v$logstdby_state;

SESSION_ID STATE
---------- -------------------------
         1  APPLYING
```

再来看这个查询当前日志应用信息的例子。

例子 4-149　查询当前日志应用信息。

```
SQL> select applied time,applied scn,mining time,mining scn
  2  from v$logstdby progress;

APPLIED TIME              APPLIED SCN MINING TIME               MINING SCN
------------------------- ----------- ------------------------- ----------
17-5 月 -2010 20:51:03          825398 17-5 月 -2010 20:51:03          825399
```

上述输出说明 APPLIED_SCN 对应的 SCN 为 825398，该 SCN 之前的所有日志数据已经应用到逻辑 Standby 数据库。MINING_SCN 的 SCN 为 825399，说明该序列号之前的日志数据已经由 MINING 引擎处理过，通常 MINING_SCN 序列号要比 APPLIED_SCN 序列号要大。

- 等待日志 Gap 状态（Waiting for gap）：该状态说明已经应用了所有可获得的重做日志数

据，但是正在等待新的日志文件，或者是可能丢失了 Redo 日志数据，出现了日志中断。

- 空闲状态（Idle）：该状态说明逻辑 Standby 数据库已经成功应用了从 Primary 数据库获得的所有 Redo 数据。如果 Primary 数据库不是很繁忙，多数情况下会处于 IDLE 状态。如下例我们查询已经启动了一段时间的逻辑 Standby 的 SQL 应用状态，如下例所示。

例子 4-150 查询 SQL 应用状态。

```
SQL> col realtime_apply for a10
SQL> col state for a10
SQL>select *
  2* from v$logstdby_state

PRIMARY_DBID SESSION_ID REALTIME_A STATE
------------ ---------- ---------- ----------
  2092838584          1      Y         APPLYING
```

4.11.4 修改 DBA_LOGSTDBY_EVENTS 视图的相关参数

我们已经知道 DBA_LOGSTDBY_EVENTS 数据字典记录了在 SQL 应用过程中发生的事件。默认该数据字典记录了 100 条事件记录，如果有更多的事件需要记录，就会覆盖掉以前的事件记录。为了记录更多的事件，我们可以修改该视图记录的事件数量，可以使用 DBMS_LOGSTDBY.APPLY_SET 过程来修改。在执行修改前，必须先停止 SQL 应用，执行 DBMS_LOGSTDBY.APPLY_SET 存储过程后，再启动 SQL 应用，如下例所示。

例子 4-151 设置 DBA_LOGSTDBY_EVENTS 字典记录的事件数。

```
# 先停止 SQL 应用
SQL> alter database stop logical standby apply;

数据库已更改。
# 修改 DBA_LOGSTDBY_EVENTS 视图记录的事件数量。
SQL> execute dbms_logstdby.apply_set('max_events_recorded','1000');

PL/SQL 过程已成功完成。
# 重启 SQL 应用。
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

我们可以通过 v$logstdby_stats 来查看当前的事件记录数，如下例所示。

例子 4-152 检查 DBA_LOGSTDBY_EVENTS 视图的事件记录数。

```
SQL> col name for a40
SQL> run
  1  select name,value
  2  from v$logstdby_stats
  3* where name='maximum events recorded'

NAME                                     VALUE
---------------------------------------- ----------------------
```

```
maximum events recorded                1000
```

使用过程 dbms_logstdby.apply_set 也可以定制要记录的事件，如在逻辑 Standby 数据库上记录已经应用的 DDL 事务事件。在使用过程 dbms_logstdby.apply_set 定制要记录的事件前，同样需要先停止 SQL 应用，如下例所示。

例子 4-153　定义 DBA_LOGSTDBY_EVENTS 视图记录的事件。

```
SQL> alter database stop logical standby apply;

数据库已更改。

SQL> execute dbms_logstdby.apply_set('record_applied_ddl','true');
PL/SQL 过程已成功完成。
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

一旦修改成功，就可以通过数据字典 v$logstdby_stats 查询该事件是否被记录，如下例所示。

例子 4-154　验证事件 record applied DDL 的记录状态。

```
SQL> select name,value
  2  from v$logstdby_stats
  3  where name='record applied DDL';

NAME                                   VALUE
---------------------------------------------------------------- -----------
record applied DDL                            Y
```

此时 VALUE 的值为 Y，说明已经启动 record applied DDL 事件了。如果没有使用存储过程 dbms_logstdby.apply_set 设置记录 record applied DDL 事件，则此处 VALUE 的值为 N。

4.11.5　逻辑 Standby 的 DDL 操作

对逻辑 Standby 数据库而言，由于逻辑 SQL 应用的诸多限制以及逻辑 Standby 数据库的功能，对逻辑 Standby 数据库做出修改限制是合理的。我们可以使用 ALTER DATABASE GUARD ALL 指令做出最严格的限制，不允许除 SYS 用户外的所有用户对 Standby 数据库进行任何修改，如删除数据库对象、向表插入数据或删除数据以及创建新的数据库对象等操作。也可以使用 ALTER DATABASE GUARD STANDBY，该指令只限制 SQL 应用所涉及的 Standby 数据库对象或数据。

但是，其他非 SYS 用户毕竟有时需要在 Standby 数据库执行 DDL 操作，如创建一个表或者在已有的表上创建索引等 DDL 操作。此时就需要解决如何在当前会话中越过数据库的保护（GUARD）的问题。下面我们演示如何在当前的用户模式下创建一个测试表 TEST，步骤如下。

01　停止 Standby 应用。

```
SQL> alter database stop logical standby apply;

数据库已更改。
```

02 关闭当前会话的数据库保护。

```
SQL> alter session disable guard;

会话已更改。
```

03 创建测试表。

```
SQL> create table test
  2  as
  3  select *
  4  from scott.emp;

表已创建。
```

04 重新开启当前会话的数据库保护

```
SQL> alter session enable guard;

会话已更改。
```

05 重新启动 SQL 应用。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

注意，我们此时创建的表 TEST 是不受 SQL 应用保护的，如果启动了 ALTER DATABASE GUARD STADNBY 指令来保护 Standby 数据库，此时具有权限的用户可以对创建的 TEST 表做任何 DML 或 DDL 操作。

4.11.6 DBMS_LOGSTDBY.SKIP 取消同步

在 Data Guard 环境下，逻辑 Standby 数据库端接收到的 Redo 数据会自动被执行并写入 Standby 的数据库。在某些情况下需要跳过某个数据库对象的操作（DML 或 DDL），此时就必须取消该数据库对象与 Primary 数据库对象的同步关系，即告诉 Standby 数据库不要应用涉及该对象的 Redo 数据。Data Guard 提供了过程 DBMS_LOGSTDBY.SKIP 来完成这个任务。下面我们就看一下该过程的用法。

先给出该过程的语法结构，再给出具体的例子做演示。

```
DBMS_LOGSTDBY.SKIP(
            Stmt                IN VARCHAR2,
            Schema_name         IN VARCHAR2 DEFAULT NULL,
            Object_name         IN VARCHAR2 DEFAULT NULL,
            Proc_name           IN VARCHAR2 DEFAULT NULL,
            Use like            IN BOOLEAN DEFAULT TRUE,
            Esc                 IN CHAR1 DEFAULT NULL);
```

在上述参数中只有 stmt 参数是必须的，其他参数根据需要再设置。参数 stmt 最常使用的是 SCHEMA_DDL 和 DML 操作，SCHEMA_DDL 操作是指模式对象的所有 DDL 操作，而 DML 是指对表的 DML 操作，比如增、删、改等操作。具体 stmt 涉及的关键字值可以参考 Oracle 的官方文

档，这里不再一一列出。

下面我们演示对 SCOTT 模式的数据库对象 EMP 表取消同步的操作步骤，无论何时该取消对象同步的操作都要先停止 Redo 应用，这里是停止 SQL 应用，然后取消表同步、重启 SQL 应用来达到取消同步的目的。步骤如下。

01 关闭 SQL 应用。

```
SQL> alter database stop logical standby apply;

数据库已更改。
```

02 取消 DDL 保护和 DML 保护。

```
SQL> execute dbms_logstdby.skip(stmt=>'schema_ddl',-
> schema_name=>'scott',-
> object_name=>'emp');

PL/SQL 过程已成功完成。
```

注意上述过程跳过对模式对象的 DDL 操作，也就是在逻辑 Standby 端取消了对模式对象的同步关系，因为 stmt=>'schema_ddl'。下面我们再次执行该过程，跳过对特定表的 DDL 操作。

```
SQL> execute dbms_logstdby.skip('dml','scott','emp');

PL/SQL 过程已成功完成。
```

03 重启 SQL 应用。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

此时，我们已经成功执行了过程 DBMS_LOGSTDBY.SKIP，取消了 SCOTT 模式对象的同步关系，也取消了 SCOTT 模式表的 DML 操作的同步关系。此时，如果在 Primary 数据库端在表 EMP 中插入一行数据，该数据不会再同步到 Standby 数据库了。

我们通过字典 dba_logstdby_skip 来查看 SCOTT 用户涉及的被取消同步的信息，如下例所示。

例子 4-155 查看 SCOTT 用户涉及的被取消同步的信息。

```
SQL>select *
  2  from dba_logstdby_skip
  3  where owner='scott';

ERROR     STATEMENT_OPT          OWNER      NAME        U  E  PROC
--------  --------------------  ----------  -----------  --------
N         SCHEMA_DDL             scott      emp         Y
N         DML                    scott      emp         Y
```

字典 dba_logstdby_skip 清晰的记录了我们取消同步的模式以及操作信息。

我们这里做一个测试，在 Primary 数据库端 SCOTT 模式的表 EMP 中删除一行数据，提交后切换日志，如下例所示。

例子 4-156 删除表 EMP 中一行数据。

```
SQL> delete from scott.emp
  2  where empno=9999;

已删除 1 行。
```

例子 4-157 提交并切换日志

```
SQL> commit;

提交完成。

SQL> alter system switch logfile;

系统已更改。
```

此时，我们转换到 Standby 数据库，查询 EMPNO=9999 的数据行是否存在，如下例所示。

例子 4-158 在 Standby 数据库查询 EMPNO=9999 的数据行。

```
SQL>select *
  2  from scott.emp
  3* where empno=9999

    EMPNO   ENAME     JOB     MGR HIREDATE      SAL    COMM     DEPTNO
------ ----- -------- ------ ---------- ------ ------ ---- ------------ ----
      9999   test      test     9999 12-1月 -99   2000              10
```

从上述输出看出，EMPNO=9999 的数据行依然存在，虽然我们已经在 Primary 数据库端删除了 EMPNO=9999 的数据行，但是因为我们取消了对 EMP 表的 DML 操作同步关系，所以造成此时的 Primary 数据库端对 EMP 表的 DML 操作不会同步到 Standby 数据库端。

4.11.7 DBMS_LOGSTDBY.UNSKIP 恢复同步

与取消对象同步相对应的是恢复对象同步，如果想将某一个已经取消同步的对象恢复同步关系，就可以使用 DBMS_LOGSTDBY.UNSKIP 过程来实现。

下面先给出该过程的语法，然后介绍恢复同步的具体步骤。

```
DBMS LOGSTDBY.UNSKIP(
            Stmt                IN VARCHAR2,
            Schema name         IN VARCHAR2 DEFAULT NULL,
               Object_name      IN VARCHAR2 DEFAULT NULL);
```

其中的关键字与 DBMS_LOGSTDBY.SKIP 的相同，因为是恢复过程，所以关键字自然具有对应关系。下面我们演示如何取消对象同步，首先需要停止 SQL 应用、UNSKIP 对象，然后重启 SQL 应用。

01 关闭 SQL 应用。

```
SQL> alter database stop logical standby apply;
```

```
数据库已更改。
```

02 取消 DDL 保护和 DML 保护。

```
SQL> execute dbms_logstdby.unskip(stmt=>'schema_ddl',-
> schema name=>'scott',-
> object_name=>'emp');

PL/SQL 过程已成功完成。
```

上面语句取消模式的 DDL 操作同步，下面我们再次执行该过程，取消 EMP 表的 DML 操作。

```
SQL> execute dbms logstdby.unskip('dml','scott','emp');

PL/SQL 过程已成功完成。
```

03 重启 SQL 应用。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

此时，我们恢复了同步关系，下面我们继续在 Standby 数据库上做查询，继续查询 EMP 表中 EMPNO=9999 的数据行是否存在，如下例所示。

例子 4-159 查询 EMP 表的 EMPNO=9999 的数据行。

```
SQL> select *
  2  from scott.emp
  3  where empno=9999;

    EMPNO ENAME     JOB      MGR HIREDATE       SAL      COMM    DEPTNO
------ ------ ----- -------- --------- ------ -------- -------------
     9999 test      test     9999   12-1月 -99  2000                    10
```

从输出发现我们恢复了同步关系，但是先前在 Primary 数据库端所做的修改，即删除了 EMPNO=9999 的数据行，该操作没有在恢复同步关系后传播到 Standby 数据库。其实，这里有一个问题，在恢复同步关系后，涉及的对象变化需要重新与 Primary 数据库做一次同步，才可以将 Primary 数据库端对该对象的操作同步到 Standby 数据库端。

我们使用 DBMS_LOGSTDBY.INSTANTIATE_TABLE 过程来重新执行同步关系。不过此时需要先在 Standby 数据库上创建到 Primary 数据库的 DBLINK，如下例所示。

例子 4-160 创建到 Primary 数据库的 DBLINK。

```
SQL> create database link linktopri connect to system identified by oracle using
'lszstdby';

数据库链接已创建。
```



当前的 Standby 数据库为 lszpri，而 Primary 数据库为 lszstdby，所以在创建到 Primary 数据库的 DBLINK 时，我们使用了 using 'lszstdby' 子句。

接着停止 Redo 应用再执行同步操作，如下面例子所示。

例子 4-161 停止 Redo 应用。

```
SQL> alter database stop logical standby apply;

数据库已更改。
```

例子 4-162 执行重新同步。

```
SQL> execute dbms_logstdby.instantiate_table('SCOTT','EMP','LINKTOPRI');

PL/SQL 过程已成功完成。
```

此时重新同步已经完成，在 Standby 数据库端我们继续查询验证同步结果，如下例所示。

例子 4-163 验证同步结果。

```
SQL>  select *
  2  from scott.emp
  3  where empno=9999;

未选定行
```

从输出结果可以看出，已经没有 EMPNO=9999 的数据行了，说明重新同步成功。我们在 Primary 数据库端删除了 EMPNO=9999 的 DML 操作，在重新同步后传播到了 Standby 数据库，又实现了与 Standby 的同步关系。下面还要重新启动 Redo 应用。

例子 4-164 重新启动 Redo 应用。

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

4.12 深入学习Redo传输服务

在 Data Guard 环境下，Redo 传输服务体现了 Data Guard 的本质。本节将从整体上介绍 Data Guard 的 Redo 传输过程，同步与异步传输的区别，以及使用 ARCn 进程或者 LGWR 进程完成 Redo 数据传输的流程。本节不会涉及具体操作，都是从原理上介绍，如果读者理解了本节介绍的内容，一定会抓住学习和使用 Data Guard 的本质内容。

无论在单实例环境，还是在 RAC 环境，只要部署了 Data Guard 都是通过 Redo 传输服务实现数据同步的。这里笔者想多谈几句自己学习的体会，就是如何理解“树木”和“森林”的关系，我们在学习开始，都喜欢简单的例子来帮助理解系统的运行，参数的修改结果等信息。但是例子只是“解渴”的功能，解一时之急，也就相当于“授人以鱼”，注意这里笔者写的是“鱼”啊☺，而原理、内部机制才是“授人以渔”，注意这里改成“渔”了☺。进入 Data Guard 的世界，如果不抓住或是理解 Redo 传输服务，就无法掌握其本质精髓，就无法在处理问题时有宏观的把控。所以，在学习中既要见“树木”，更要见“森林”，知道树木则可以从用的角度实现细节操作，知道“森林”则可以从更高的视角认识和把控全局。二者不可或缺，但是在学习中重点在心中有森林，再通过小

例子或具体的演示验证对原理机制的理解。

学习方法因人而异，笔者不想强求，如果读者觉得通过示例来得快就先看例子，很多内容如 EXP、IMP 数据库的导入导出通过例子就很容易把握。但是仅仅通过例子学习，不看重或充分理解原理，你慢慢会发现你可以把控的东西很少，没有畅快淋漓的感觉，有的只是机械的运用，想必这个时候，你会把注意力转移到学习原理和机制上来了。从实践到理论，再从理论到实践是一个循环往复的提高过程，在这个过程中你的把控能力也自然得到提高。

好了，下面我们正式进入 Redo 传输机制的学习。在 Primary 数据库端有两种方式来发送 Redo 数据，一种是通过 ARCn 进程，一种是通过 LGWR 进程，我们以这两种发送 Redo 数据的方式来解释 Redo 传输的本质。

4.12.1 通过 ARCn 进程来传送 Redo

图 4-8 是 Primary 数据库端使用 ARCn 进程来传输重做日志到 Standby 端的示意图，下面我们分析整个 Redo 传输流程，以及在传输过程中由谁触发 Redo 传输以及归档位置等信息。同时，为了解释清楚，我们先分析整个流程，再按照 Primary 数据库端和 Standby 数据库端来分别介绍涉及的多个参数。

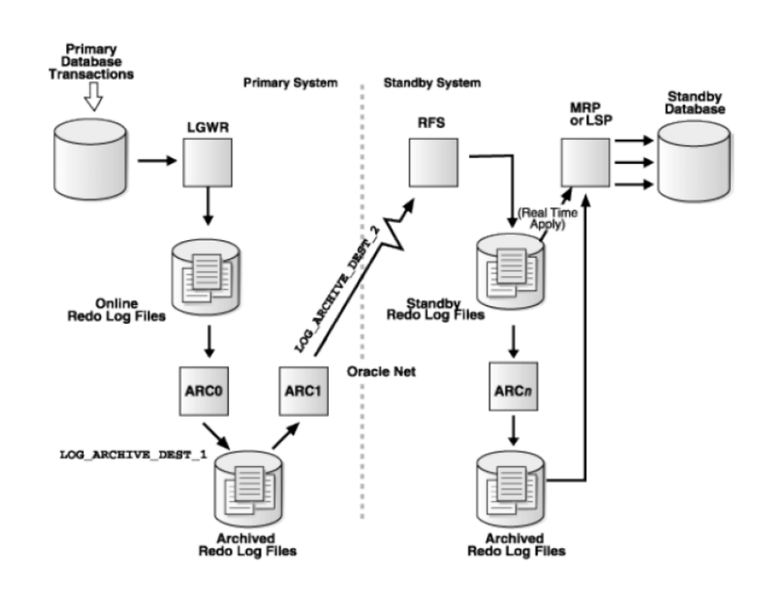


图 4-8 ARCn 归档进程传送 Redo 数据

在 Primary 数据库，一旦使用 ARCn 进程来传输 Redo 数据，只能采用异步传输的方式。在 Primary 数据库端 LWGR 进程将数据库高速缓冲区中的相关数据写入在线重做日志文件，当发生日志切换或者其他触发归档的事件时，ARC0 归档进程就会实时对 Redo 数据归档，一旦归档完毕就会触发 ARC1 进程将 Redo 数据传输到远端数据库。

在 Standby 数据库端，RFS（Remote File Server）进程接收从 Primary 数据库端传输过来的 Redo 数据，此时我们分两种情况，一种是启动了实时应用，一种是没有启动实时应用。

如果启动了实时应用，Redo 数据写入 Standby 重做日志时，会同时使用 MRP 进程或 LSP 进

程将 Redo 数据写入 Standby 数据库。此时的 Standby 重做日志数据依据触发条件启动归档，将 Redo 数据写入归档日志文件。

如果没有启动实时应用，此时 RFS 进程将 Redo 数据写入 Standby 重做日志文件，该步骤完成后 ARCn 进程会接着被启动并写入 Standby 数据库端的归档日志文件，然后再由 MRP 进程或 LSP 进程将 Redo 数据写入 Standby 数据库。



使用 ARCn 进程传输 Redo 数据是 Data Guard 默认的方式，又因为它只能使用异步传输的方式，所以不能提高诸如最大保护和最高可用性的保护方式。

但是必须注意，上述只是自然语言描述，目的是使得读者对上述过程有一个清楚地理解，然后我们再从实际配置的角度分析这个过程是通过哪些参数配置实现的。

1. 在 Primary 数据库端配置

在 Primary 数据库端需要配置 log_archive_dest_n 参数来指定本地日志归档目录，以及如何向 Standby 数据库传送 Redo 数据。创建逻辑 Standby 时 Primary 数据库端的参数设置，如下例所示。

```
#参数 log_archive_dest_1 用于将本地重做日志数据写入本地目录存储
log_archive_dest_1='location=e:\oracle\product\10.2.0\oradata\lszpri
valid_for=(online_logfiles,all_roles) db_unique_name=lszpri'
log_archive_dest_state_1=enable

#参数 log_archive_dest_2 用于将本地重做日志数据异步传送到远端 Standby 数据库
log_archive_dest_2='service=lszstdby async
valid_for=(online_logfiles,primary_role) db_unique_name=lszstdby'
log_archive_dest_state_2=enable
```

其中参数 log_archive_dest_1 的 location 属性说明本地归档日志的存储目录，valid_for 属性说明将在线重做日志写入归档目录，属性 db_unique_name 说明与在线重做日志对应的数据库，即参数 log_archive_dest_1 的含义是将数据库 lszpri 的在线重做日志数据写入本地目录 e:\oracle\product\10.2.0\oradata\lszpri 下。

同时需要配置参数 log_archive_dest_2，该参数主要说明如何将本地 Redo 数据传输到 Standby 数据库。属性 service=lszstdby 说明使用 Oracle Net 服务通过套接字连接与 Standby 数据库进行通信，valid_for=(online_logfiles,Primary_role)属性说明当前的数据库为 Primary 角色时，将在线重做日志的数据传输到 Standby 数据库，属性 db_unique_name=lszstdby 说明 Primary 数据库将 Redo 数据通过 service 指定的属性值连接到 Standby 端对应的数据库，因为一个 Standby 主机上可以安装多个数据库，而且多个数据库的 service 参数可以设置相同，所以就由 db_unique_name 指定对应的数据库。



此时我们在 log_archive_dest_2 参数中没有指定 ARCn 或者 LGWR 进程传输 Redo 数据，此时默认使用 ARCn 进程传输 Redo 数据。

参数 log_archive_dest_state_1 说明是否应用当前的归档目录，其参数必须设置为 ENABLE 才会启动该归档目录。

2．在 Standby 数据库端配置

在 Standby 数据库端需要设置重做日志的存储目录，即将从 Primary 数据库传输过来的 Redo 数据归档存放的目录，同样适用于 log_archive_dest_n 参数设置，如下例所示。

例子 4-165 在 Standby 数据库端 Redo 传输参数设置。

```
log_archive_dest_1='location= E:\oracle\product\10.2.0\oradata\lszstdby
valid_for=(online_logfiles,all_roles) db_unique_name=lszstdby'
log_archive_dest_state_1=enable

log_archive_dest_3='location=e:\oracle\product\10.2.0\oradata\lszstdby_1
valid_for=(standby_logfiles,standby_role) db_unique_name=lszstdby'
log_archive_dest_state_3=enable
```

在上例中，如果是逻辑 Standby 数据库则必须设置参数 log_archive_dest_3，该参数说明当前的数据库为 Standby 角色时，将 Standby 日志文件归档到 location 指定的目录下。而参数 log_archive_dest_1 说明本地的重做日志数据的归档路径。

3．在 Standby 端启动实时传输

实时传输是指当 RFS 进程接收 Redo 数据并写入 Standby 重做日志文件时，同时将 Redo 数据写入 Standby 数据库，写入方式根据逻辑 Standby 与物理 Standby 分别使用 SQL 应用或者 Redo 应用完成数据的写入。

在逻辑 Standby 与物理 Standby 端启动实时传输的指令不同，下面我们分别介绍这两种情况启动实时传输的方式，该操作在 Standby 端启动，如下面例子所示。

例子 4-166 在物理 Standby 数据库端启动实时应用。

```
SQL>alter database recover managed standby database using current logfile;

数据库已更改。
```

例子 4-167 取消物理 Standby 数据库端启动实时应用。

```
SQL>alter database recover managed standby database cancel;

数据库已更改。
```

例子 4-168 在逻辑 Standby 数据库端启动实时应用。

```
SQL>alter database start logcial standby apply immediate

数据库已更改。
```

例子 4-169 在逻辑 Standby 数据库端启动实时应用。

```
SQL> alter database stop logical standby apply;

数据库已更改。
```

4.12.2 LGWR 进程同步传送 Redo

图 4-9 是使用 LGWR 进程同步传输 Redo 数据的过程，该图清晰地给出了 Redo 数据传输的过程，以及传输过程中涉及的数据库组件和进程。

首先，用户事务将变化写入数据库高速缓存，满足一定条件后启动 LGWR 进程将 Redo 数据写入本地在线的重做日志文件，同时 LGWR 进程启动了 LNSn（LGWR Network Server Process）进程把 Redo 数据传输到 Standby 数据库端，在 Standby 数据库端的 RFS 进程接收 Redo 数据并写入 Standby 重做日志文件。如果此时启动了实时应用，则使用 SQL 应用或 Redo 应用将 Redo 数据应用到 Standby 数据库，如果没有启动实时应用，则将 Redo 数据写入 Standby 重做日志文件。在 Redo 数据写入 Standby 数据库前 Primary 数据库端的事务不会结束，这也是同步的含义。

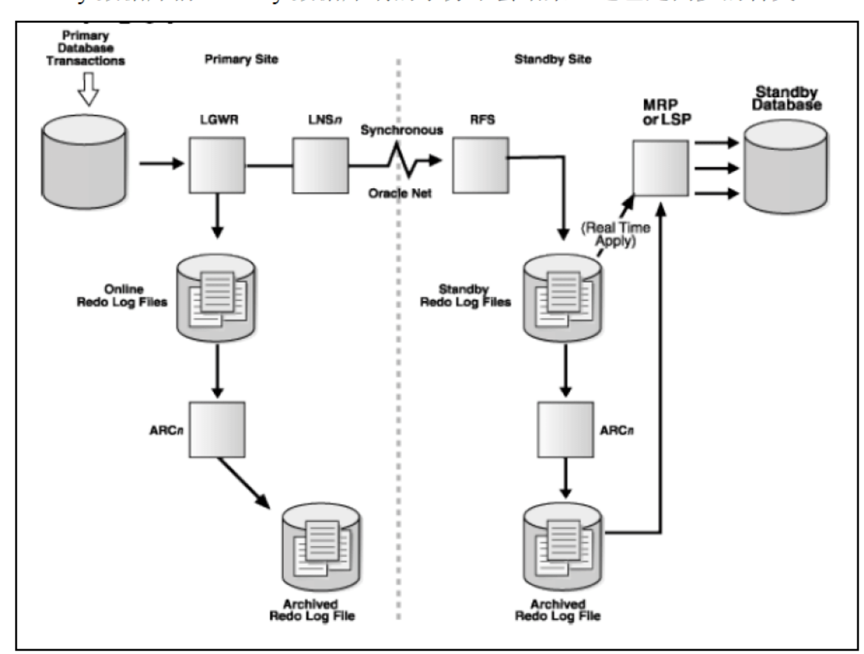


图 4-9 LGWR 日志写进程传送同步 Redo 数据

对于在 Primary 数据库端以及 Standby 数据库端的参数设置和使用 ARCn 进程设置的参数类似，关键是在 Primary 数据库端将 log_archive_dest_n 参数的属性增加 sync 关键字，告诉 Primary 数据库在传输日志时启动同步传输，如下例所示。

```
#参数 log archive dest 2 用于将本地重做日志数据异步传送到远端 Standby 数据库
log archive dest 2='service=lszstdby lgwr sync
valid for=(online logfiles,primary role) db unique name=lszstdby'
log_archive_dest_state_2=enable
```

4.12.3 LGWR 进程异步传送 Redo

图 4-10 是使用 LGWR 进程异步传输 Redo 数据的过程，该图清晰地给出了 Redo 数据传输的过程，以及传输过程中涉及的数据库组件和进程。在某些情况下，比如 Primary 数据库端有长事务

执行，此时就会出现因为传输 Redo 数据没有结束而造成的等待事件。

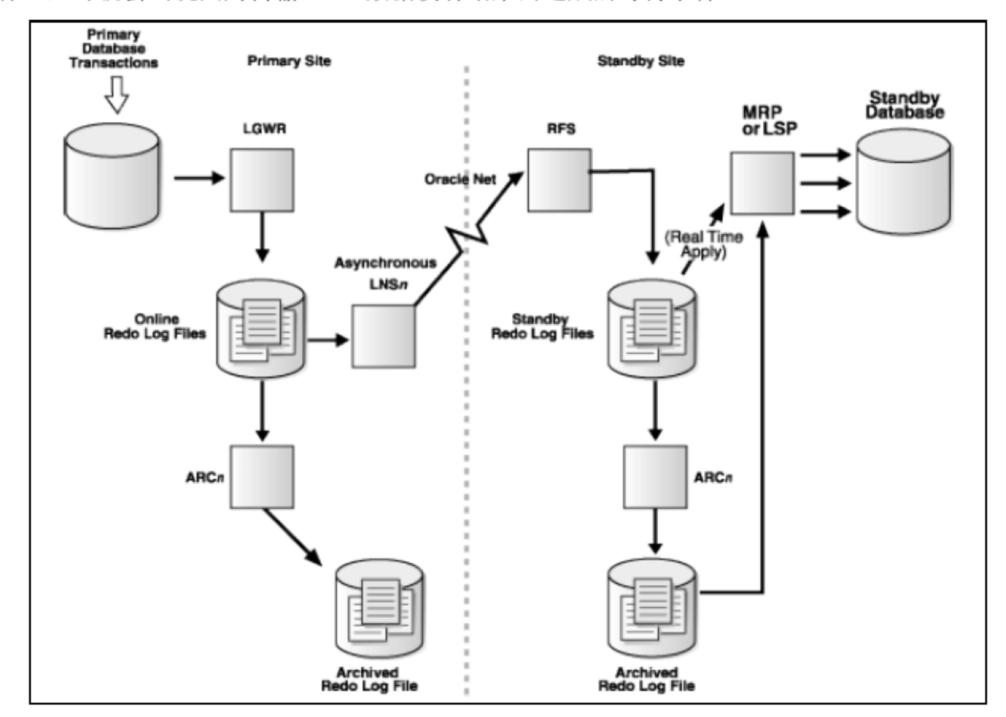


图 4-10　LGWR 日志写进程传送同步 Redo 数据

LGWR 进程将 Redo 数据写入 Primary 数据库端的在线重做日志，然后事务结束，Primary 数据库端的 LNSn 进程会从在线日志文件读取 Redo 数据，并经过网络传输到 Standby 数据库。Standby 数据库端的 RFS 进程获得 Redo 数据并写到 Standby 重做日志，此时如果启动了实时应用，就同时将 Redo 数据写入 Standby 数据库。如果没有启动实时应用，则 RFS 进程将 Redo 数据写入 Standby 重做日志，然后 ARCn 进程将 Standby 重做日志数据写入归档目录。

4.13　使用RMAN创建Standby

RMAN 是 Oralce 的数据库备份和恢复工具。在前面的章节我们讲述了如何创建物理 Standby 和逻辑 Standby 数据库，这些都是基于手动对 Primary 数据库文件的备份。而 RMAN 创建 Standby 数据库却不需要手工备份数据库文件，而是通过 RMAN 自己对数据库文件的备份实现自动化创建 Standby 数据库。

使用 RMAN 创建 Standby 数据库的优点如下：

- RMAN 可以通过预先备份的 Primary 数据库备份文件创建 Standby 数据库，而不影响 Primary 数据库的运行。
- RMAN 自动命名数据库文件以及文件目录。
- RMAN 可以从备份中复原归档日志并进行介质恢复，实现与 Primary 数据库的同步。

4.13.1 RMAN 创建 Standby 数据库的前提

01 在使用 RMAN 创建 Standby 数据库前需要 Primary 数据库备份，这是创建的数据基础。

02 配置 Standby 数据库实例。

03 要修改 Primary 数据库端以及 Standby 数据库端的初始化参数文件，修改 Primary 数据库端初始化参数使得它可以像新创建的 Standby 数据库发送 Redo 数据，修改 Standby 数据库端的初始化参数使得它做好加入 Data Guard 的准备。

04 启动监听创建网络服务（接收来自 Primary 数据库的 Redo 数据）。

05 启动数据库到 nomount 状态。

06 通过 RMAN 创建 Standby 控制文件并将该控制文件拷贝到正确的目录下，并修改相应的控制文件名使得其与 Standby 库初始化参数文件中的控制文件路径及名称相同。

07 选择是否使用重命名策略。

重命名策略是将 Primary 数据库的数据库文件重命名为一个新的数据库文件名，当 Primary 数据库与 Standby 数据库要求不同的目录结构时使用重命名策略。如果 Primary 数据库与 Standby 数据库在不同主机且存储目录相同则不需要重命名策略，RMAN 自动将 Primary 数据库的数据库文件拷贝到 Standby 数据库的相同目录下（前提是必须已经创建了相关目录）。

下面进一步说明是否采用重命名策略的影响。如果不采用重命名策略，即在 Primary 数据库与 Standby 数据库端采用一样的目录结构，此时并不需要设置初始化参数 db_file_name_convert 以及 log_file_name_convert，但是需要在使用 DUPLICATE 创建 Standby 数据库时指定 NOFILENAMECHECK 参数，以阻止在 Standby 数据库端因为该目录名与原数据库文件目录名相同而报错，这个参数的创建指令如下例所示。

```
CONFIGURE TARGET DATABASE FOR STANDBY NOFILENAMECHECK DORECOVER
```

如果采用重命名策略，则需要重新设置数据库文件的逻辑位置。此时有多种修改方式，最常用的就是修改初始化文件参数 db_file_name_convert 以及 log_file_name_convert 的值。也可以使用 SET NEWNAME 指令，如下例所示。

```
SET NEWNAME FOR DATAFILE 1 TO '/ORACLE/ORADATA/SYSTEM01.DBF'
```

或者也可以使用 CONFIGURE AUXNAME 指令定义 Standby 数据库端的库文件目录，如下例所示。

```
CONFIGURE AUXNAME FOR DATAFILE 1 TO  '/ORACLE/ORADATA/SYSTEM01.DBF'
```

最后就可以通过如下指令创建 Standby 数据库。

```
DUPLICATE TARGET DATABASE FOR STANDBY
```

若需要在创建过程中恢复数据库，则可以输入如下指令。

```
DUPLICATE TARGET DATABASE FOR STANDBY DORECOVER
```

4.13.2 RMAN 创建 Standby 数据库实例

本节我们通过实例演示使用 RMAN 创建物理 Standby 数据库的具体步骤，以及创建过程中的

注意事项。实例的创建环境是我们在同一台计算机上的不同目录下创建 Standby 数据库，并且是使用 RMAN 备份来创建的。

01 创建 Standby 实例。

因为我们在 Windows 环境下创建的实例，所以需要使用 ORADIM 工具，如果在其他系统下不需要这个步骤。如下例所示在 Windows 系统上创建 Standby 数据库实例。

```
D:\>oradim -new -sid rmanstdby
实例已创建。
```

02 启动监听并配置网络服务名。

在图 4-11 所示的窗体中使用 DBCA 在 Primary 数据库端启动创建网络服务，在图 4-12 所示的窗体中设置新的网络服务名为 rmanstandby。



图 4-11　添加网络服务名



图 4-12　设置网络服务名参数

在创建完网络服务之后，就可以通过 Oracle Net 传输数据到 Standby 数据库，我们使用 tnsping 指令测试网络服务是否正常，如下例所示。

例子 4-170 测试网络服务是否正常。

```
    D:\>tnsping rmanstdby

    TNS Ping Utility for 32-bit Windows: Version 10.2.0.1.0 - Production on 19-5
月 -2010 16:07:12

    Copyright (c) 1997, 2005, Oracle.  All rights reserved.

    已使用的参数文件:


    已使用 TNSNAMES 适配器来解析别名
    Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL =
    TCP)(HOST = 127.0.0.1)(PORT = 1521))) (CONNE
    CT_DATA = (SERVICE_NAME = rmanstdby)))
    OK (20 毫秒)
```

上述操作时在 Standby 数据库上执行，从测试返回结果可以看出网络服务运行正常。

03 创建密码文件，如下例所示。

例子 4-171 创建密码文件。

```
    D:\>orapwd     file=e:\oracle\product\10.2.0\db_1\database\PWDrmanstdby.ora
password=oracle
    entries=30
```

04 创建 Standby 数据库端的初始化参数。

创建初始化参数文件方式很多，可以根据 init.ora 文件模板来创建，但是最简单的方式还是从 Primary 数据库创建一个 pfile 然后修改相关目录，最后再通过该 pfile 创建 Standby 数据库的 spfile 参数文件。我们首先创建一个 pfile 文件，如下例所示。

例子 4-172 创建 spfile 文件。

```
    SQL> create pfile='e:\oracle\product\10.2.0\db_1\
        database\INITrmanstdby.ora' from spfile;

    文件已创建。
```

下面我们修改参数文件 INITrmanstdby.ora 文件，修改结果如下例所示。

```
    *.audit_file_dest='E:\oracle\product\10.2.0/admin/rmanstdby/adump'
    *.background_dump_dest='E:\oracle\product\10.2.0/admin/rmanstdby/bdump'
    *.compatible='10.2.0.1.0'
    *.control_files='E:\oracle\product\10.2.0
    \oradata\rmanstdby\control01.ctl','E:\oracle\product\10.2.0
    \oradata\rmanstdby\control02.ctl','E:\oracle\product\10.2.0
    \oradata\rmanstdby\control03.ctl'
    *.core_dump_dest='E:\oracle\product\10.2.0/admin/rmanstdby/cdump'
    *.db_block_size=8192
    *.db_domain=''
    *.db_file_multiblock_read_count=16
    *.db_file_name_convert='oradata\lszstdby','oradata\rmanstdby'
```

```
    *.db_name='LSZSTDBY'#db_name
    *.db_recovery_file_dest='E:\oracle\product\10.2.0/flash_recovery_areaforst
dby'
    *.db_recovery_file_dest_size=2147483648
    *.db_unique_name='lszstdby'
    *.dispatchers='(PROTOCOL=TCP) (SERVICE=rmanstdbyXDB)'
    *.job_queue_processes=10
    *.log_archive_config='dg_config=(lszpri,lszstdby,rmanstdby)'
    *.log archive dest 1='location=e:\oracle\product\10.2.0
    \oradata\rmanstdby                    valid_for=(online_logfiles,all_roles)
db unique name=rmanstdby'
    *.log archive dest state 1='enable'
    *.log archive format='log%t %s %r.arc'
    *.log file name convert='oradata\lszstdby','oradata\rmanstdby'
    *.standby file management='auto'
    *.undo management='AUTO'
    *.user_dump_dest='E:\oracle\product\10.2.0/admin/rmanstdby/udump'
```

上述加粗字体为修改后的内容，这里关键是修改文件目录，接下来在 Standby 数据库端创建 SPFILE。

例子 4-173 创建 SPFILE。

```
    D:\>set oracle sid=rmanstdby

    D:\>sqlplus /nolog

    SQL*Plus: Release 10.2.0.1.0 - Production on 星期三 5月 19 16:50:26 2010

    Copyright (c) 1982, 2005, Oracle.  All rights reserved.

    SQL> conn sys/oracle as sysdba
    已连接到空闲例程。
    SQL> create spfile from pfile;

    文件已创建。
```

05 启动数据库到 nomount 状态，如下例所示。

例子 4-174 启动数据库。

```
    SQL> startup nomount;
    ORACLE 例程已经启动。

    Total System Global Area     603979776 bytes
    Fixed Size                   1250380 bytes
    Variable Size                163580852 bytes
    Database Buffers             432013312 bytes
    Redo Buffers                 7135232 bytes
```

06 从 Primary 数据库创建 Standby 数据库控制文件，如下例所示。

例子 4-175 创建 Standby 数据库控制文件。

```
RMAN> connect target sys/oracle

已连接到目标数据库: LSZSTDBY (DBID=2092838584, 未打开)

RMAN> copy current controlfile for standby to 'e:\oracle\product\
    10.2.0\oradata\rmanstdby\control01.ctl';

启动 backup 于 19-5月 -10
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=156 devtype=DISK
通道 ORA_DISK_1: 启动数据文件副本
复制备用控制文件
输出文件名 = E:\ORACLE\PRODUCT\10.2.0\ORADATA\RMANSTDBY\CONTROL01.CTL 标记 =
TAG20100519T174702 recid = 1 时间戳 = 71943
0423
通道 ORA_DISK_1: 数据文件复制完毕, 经过时间: 00:00:01
完成 backup 于 19-5月 -10

RMAN>
```

这里也可以采用其他方法来创建 Standby 数据库的控制文件，如使用 alter database create standby controlfile as 'e:\oracle\standbyctl.ctl'语句。

然后我们切换到 Standby 数据库到 MOUNT 状态。

07 切换 Standby 数据库到 MOUNT 状态，如下例所示

```
SQL> alter database mount;

数据库已更改。
```

08 使用 RMAN 从 Primary 数据库连接到 Standby 数据库，如下例所示。

例子 4-176 使用 RMAN 连接到 Standby 数据库。

```
D:\>set oralce sid=lszstdby
D:\>rman

恢复管理器: Release 10.2.0.1.0 - Production on 星期三 5月 19 17:50:52 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


RMAN> connect auxiliary sys/oracle@rmanstdby

已连接到辅助数据库: LSZSTDBY (DBID=2092838584, 未打开)
```

在这个步骤中读者可以想到，因为要将 Primary 数据库的数据文件拷贝到 Standby 数据库上去，所以需要建立一个通信方式，这里就是通过 RMAN 连接建立数据传输的通信方式。

09 备份 Primary 数据库，如下例所示。

例子 4-177　使用 RMAN 备份 Primary 数据库。

```
RMAN> backup database;
```

这个步骤在生产环境下没有问题，但是在测试环境下有时会忘记备份 Primary 数据库，切记我们是依据 RMAN 备份的 Primary 数据库文件创建逻辑 Standby。

10 创建 Standby 数据库，如下例所示。

例子 4-178　通过 RMAN 创建 Standby 数据库。

```
RMAN> connect target sys/oracle@rmanstdby

已连接到目标数据库: LSZSTDBY (DBID=2092838584, 未打开)

RMAN> duplicate target database for standby;
```

这个步骤很简单，一旦执行 duplicate target database for standby 语句，整个过程根本不需要人工干预，RMAN 自动完成并把数据库启动到 MOUNT 状态。

4.14　RAC环境下创建物理Standby

在 RAC 环境下，创建 Standby 数据库其实和单实例环境下的创建过程几乎一样，但是基于 RAC 环境的特殊性，如多个 Primary 数据库实例、保证多实例访问的共享存储特性等，这些特性要求创建 RAC 环境下的 Standby 数据库必须注意一些事项。

本节介绍这些注意事项。

1．在 Primary 数据库端

配置 RAC 环境下的每个 Primary 数据库初始化参数，如文件名转换、本地归档目录及相关属性以及负责传输到远端 Standby 数据库的 log_archive_dest_n 参数及属性等，读者可以参考创建物理和逻辑 Standby 数据库的参数设置。

此时注意在 log_archive_dest_n 中的 SERVICE 属性必须是 Standby 数据库的网络服务名，在设置之前要使用 tnsping 测试网络服务名是否正确。

2．在 Standby 数据库端

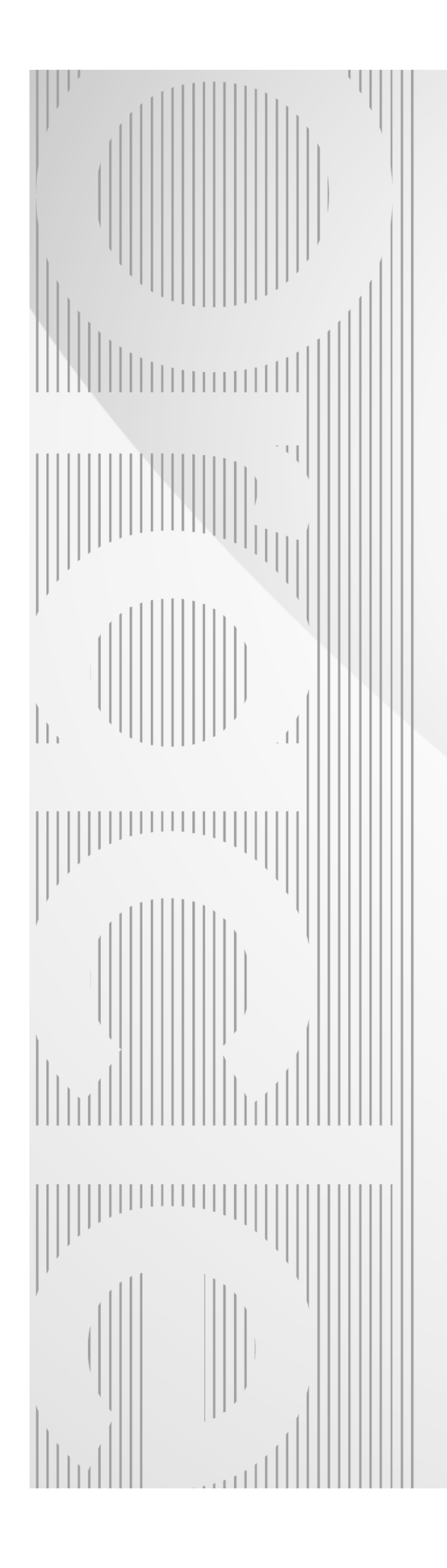
在逻辑 Standby 数据库上创建 Standby 日志文件，并要求这些文件创建在共享存储上以保持 RAC 的多个逻辑 Standby 数据库都可以访问，如集群文件或者 ASM 文件。

在每个 Standby 数据库上配置 Standby 日志的归档目录，必须保证该归档目录创建在共享存储上，以使得多个 Standby 数据库实例共享访问。也就是要求每个 Standby 数据库的 Standby 日志归档目录相同。

4.15　本章小结

本章我们重点讲了 Data Guard 的实现原理以及实务操作。在 Data Guard 实施中要充分理解逻

辑 Standby 和物理 Standby 数据库，Primary 数据库和 Standby 数据库以及角色的概念。接着我们通过具体的例子演示如何创建物理 Standby 数据库，以及如何管理物理 Standby 数据库。读者最好根据书中的步骤亲自操作一次，对深入理解物理 Standby 数据库很有好处。对于逻辑 Standby 的创建和维护采用了与物理 Standby 相同的描述方式，读者最好也亲自实践一番。逻辑 Standby 主要是使用 SQL 应用完成 Redo 数据的拷贝，在逻辑 Standby 中有些 SQL 语句是不允许执行的，这点读者要注意。为了深入理解 Data Guard 的原理，我们引入深入学习 Redo 传输服务一节，通过对其内容的学习读者可以从本质上理解各种 Redo 传输机制以及 Redo 传输的详细过程。



·第2部分·

数据库优化

本部分包括第 5～7 章，主要介绍数据库优化。数据库优化是一个十分复杂的系统行为，本书将优化组件进行分类，分别介绍不同的优化原则以及具体的优化方法。希望读者再结合自己的工作经验“大胆假设，小心求证”，相信在优化之路上可以走得顺畅些。

第 5 章

SQL优化

SQL 优化是 Oracle 数据库优化的重要内容，在 Oracle 数据库性能问题中，SQL 的优化占有重要的地位，尤其在数据库系统的设计阶段、编码阶段，高效的 SQL 语句可以极大减少系统运行之后的性能问题，如选择并创建合适的索引、使用合理的表连接以及使用绑定变量等。本章我们先分析 SQL 语句的执行过程，然后分析 Oracle 自身的 CBO 优化，即基于成本的优化方法。Oracle 为了自动优化 SQL 语句，需要各种统计数据作为优化基础，所以随后将会介绍 CBO 优化需要的统计数据及如何实现这些数据的统计。除了自动的 CBO 优化，DBA 如果参与设计以及编码，则需要主动地采取 SQL 优化方法；在系统部署之后，程序已经打包，此时就需要采取被动的优化方法，其实被动与主动是对数据库系统的不同阶段而言，都是为了减少 SQL 语句带来的性能影响。

5.1 性能调整方法

对于 Oracle 数据库系统，有关性能问题的出现往往是在系统部署一段时间以后，即大量用户开始使用该系统，系统的数据处理量和各种计算的复杂性增加时。而此时性能问题的出现，往往可以追溯到系统的初始设计阶段（当然不是绝对）。所以，为了避免频繁的性能调整，最好在初始阶段就做出合理的设计，在编码阶段就应该编写高效的 SQL 语句。这样在数据库应用系统的源头减少了性能问题出现的机率。这种在数据库设计初期以预防为目的的技能优化行为称为主动的性能调整。

而一旦系统部署之后，应用系统本身无法改变，只有对于系统的生存环境做出合理的调整以满足性能需求，如调整系统 SGA 的内存参数、调整初始化参数文件中的参数、对文件存储做合理分布以减少 I/O、对表创建合理的索引等，这样的系统性能优化行为称为被动的性能调整。其实，作为运行维护任务的 DBA 绝大多数的行为是被动的性能调整。

本章我们介绍 SQL 优化，充分理解 SQL 查询的内部过程，以及基于成本（CBO）的优化原理。这些对于编码阶段的系统可以提高 SQL 代码质量，避免低效的、严重影响系统性能的代码出现，而对于已经运行的系统，则可以最大程度上减少低效代码带来的性能影响，并在出现性能问题时容易发现造成性能瓶颈的 SQL 语句，再通过添加合适索引、建立分区表等方法来最大程度减少对系统性能的影响。

5.2 SQL查询处理过程详解

查询处理与查询优化是两个相关联的概念。查询处理时执行 SQL 语句获得数据的过程，而查询优化是通过分析 SQL 语句以及其他资源获得最佳执行计划的过程。最佳的执行计划是指消耗资源最少的执行计划，这些资源包括数据库服务器的 CPU 和系统 I/O。我们先来看一个 SQL 语句查询处理的全过程，按照不同的处理阶段，这个过程可以分为三个阶段即：语法分析阶段、语句优化阶段和语句执行阶段。

5.2.1 语法分析

语法分析在 SGA 中完成。在语法分析过程中，SQL 语句被分解为关系代数查询，通过分析这个关系代数查询来验证 SQL 语句的语法和语义是否正确，如 SQL 语句中的关键字是否正确、涉及的表定义以及相关的列是否存在、是否具有对该表的 SQL 语句中列的操作权限等，当然这些是通过数据字典完成的（数据字典存储在共享池中）。如果这些检查都顺利通过，则该 SQL 语句的分析完成，此时进入第二个阶段即获得最佳执行计划阶段。

5.2.2 语句优化

在该阶段 Oracle 默认使用基于成本的优化程序（CBO）来选择最好的执行计划。这个“最好”的标准就是消耗系统资源诸如 CPU 以及 I/O 资源最少。其间 CBO 使用系统收集的统计数据做出判断，至于如何获得这些统计数据，我们在基于成本的优化一节中再详细介绍。

在 SQL 语句优化阶段，Oracle 将语法分析树转换为一个逻辑查询，然后将逻辑查询转换为物理查询计划，此时的物理查询计划往往不只一种，因为优化器会生成几个有效的查询计划，然后会针对具体的操作对象如表或索引对每个物理查询计划做出成本消耗评估。只有消耗诸如 CPU、内存以及 I/O 最小的执行计划才被确定为最佳执行计划，该计划由查询引擎执行，此时就进入查询执行阶段。

确定最佳物理查询计划前，Oracle 会考虑如下一些因素：

- 查询中涉及的联接操作以及联接顺序；
- 操作执行的算法；
- 数据读取的最佳方式，如磁盘读取还是内存读取；
- 查询中各操作之间的数据传递方式。

5.2.3 查询执行

查询执行是整个查询过程中最简单的行为，因为此时已经形成了一个有效的物理查询计划，只要执行该计划即可，然后查询需要的数据返回给用户。如果是其他操作如 INSERT 则是插入数据，此时就是查询引擎执行查询计划的过程。

在整个查询处理过程中，最重要的是第二个阶段即优化阶段。优化过程中，优化器会考虑各种数据访问方法、表的联结顺序，以及消耗资源等问题。如果在编写 SQL 语句时能从优化器的角

度去考虑高效的 SQL 语句，则往往可以避免出现效率低下、消耗系统资源的 SQL 语句。下面将详细讲解 CBO 的原理，以获得优化 SQL 的必要知识。

5.3　基于成本的优化（CBO）

基于成本的优化即 CBO（Cost-Based Optimizer），CBO 在最终对 SQL 语句生成执行计划前，需要经历一个过程。在这个过程中需要使用优化 SQL 语句的各种数据资源，如表自身的数据，索引统计数据、系统的 I/O 速率、SQL 语句的 CPU 执行周期等，这些资源的访问或执行都需要转换成时间成本。CBO 将在几个可选的执行计划中，选择时间成本最低的一个作为该 SQL 语句的最佳执行计划。

更具体地说，CBO 使用表和索引的统计数据、SQL 语句中表和列的连接顺序、可用的索引，以及统计的操作系统数据作为依据来选择访问数据的最佳方法，整个处理过程如图 5-1 所示。

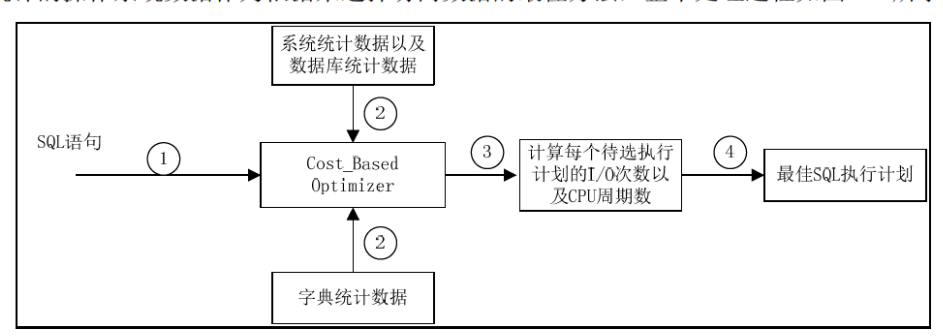


图 5-1　SQL 语句的 CBO 优化过程图

5.3.1　选择 CBO 的优化方式

优化方式的含义是为满足 SQL 优化的目标而选择的优化方式。默认条件下，CBO 将 SQL 语句的吞吐量作为优化目标。而在一些条件下，需要将响应时间作为优化目标，此时需要返回 SQL 查询的第一行或者前几行数据。

Oracle 提供了三种优化方式来满足不同的查询需求，即 ALL_ROWS、FIRST_ROWS_n 以及 FIRST_ROWS，下面分别介绍。

（1）ALL_ROWS：该优化方式是 Oracle 的默认方式，优化目标是实现查询的最大吞吐量。

（2）FIRST_ROWS_n：该优化方式使用 CBO 的成本优化输出查询的前 n 行数据，目标是以满足快速响应的查询需求。这里 n 是整数。

（3）FIRST_ROWS：该方式是 FIRST_ROWS_n 优化方式的老版本，作用是使用 CBO 的成本优化尽快输出查询的前几行数据，满足最小响应时间的需求。

对于 CBO 的优化方式，Oracle 提供在不同级别上的优化，即实例级、会话级和语句级。在设置不同级别的 CBO 优化方式之前，我们先查询当前数据库的默认优化方式，如下例所示。

例子 5-1 查询当前数据库的 CBO 优化方式。

```
SQL> show parameter optimizer_mode;

NAME                             TYPE        VALUE
-------------------------------- ----------- -------------
optimizer_mode                   string       ALL_ROWS
```

从输出看到，Oracle 默认的优化方式是 ALL_ROWS，以满足最大吞吐量为目标的优化。下面介绍如何设置 CBO 的优化方式，如下例所示。

例子 5-2 在实例级设置优化方式。

```
SQL> alter system set optimizer_mode=first_rows_10 scope=both;

系统已更改。
```



在实例级修改 CBO 的优化方式也可以在初始化参数文件中设置参数的方式，如在初始化参数文件中添加 optimizer_mode=first_rows_10。

例子 5-3 在会话级设置 CBO 优化方式。

```
SQL> alter session set optimizer_mode=all_rows;

会话已更改。
```

在会话级上设置优化方式必须使用 hint 提示，这种方式只用于查询语句，或者子查询语句，语法为 SELECT 后紧跟/*+RULE*/的方式，如下例所示。

例子 5-4 会话级上设置优化方式必须使用 hint 提示。

```
SQL>select /*+first_rows_10*/ ename,sal,mgr
2    from scott.emp;
```

这里，我们采用提高响应时间的优化原则，查询结果的前 10 行要优先输出，对于实时显示要求较高的查询语句可以采用这种方式。当然也可以选择 all_rows、first_rows 等优化方式。

5.3.2 优化器工作过程

CBO 通过 SQL 转换、确定访问路径，确定连接方法以及确定连接次序四个步骤完成 SQL 的优化，最终生成一个成本最小的执行计划。下面我们详细介绍这四个步骤。

01 根据统计数据转换 SQL 语句。

在 CBO 优化中，一个 SQL 语句往往被转换成另一种表达形式，这个转换的基础是 CBO 认为转换后的查询会更有效。比如将具有索引的 SQL 语句忽略索引的使用，将 OR 条件查询转换成 UNION ALL，将 BETWEEN 转换为>=和<=等。

02 根据资源情况选择访问路径。

一个 SQL 查询中对数据的访问的路径要根据访问这些数据所消耗的资源来判断，在多个查询路径中选择计算成本最小的一个。

03 根据统计数据选择连接方法。

在 SQL 语句中涉及多个表时，CBO 会根据统计数据以及表的主键信息来选择连接方法，在多个连接方法中选择计算成本最低的一个作为最佳连接方法。

04 确定连接次序。

连接次序是在选择了连接方法之后进行的，CBO 根据连接所涉及的数据行的数目来确定最好的连接次序。CBO 会对不同的连接次序进行计算，以选择最好的执行计划。

5.4　自动统计数据

Oracle 在默认情况下，自动运行 GATHER_STATS_JOB 作业来为表获得统计数据，并且监控所有数据库对象的所有 DML 操作。如果某个对象的统计数据自上次统计以来有超过 10% 的数据被更改，则会再次统计该对象的统计数据。下面我们查看 GATHER_STATS_JOB 的当前运行状态。

例子 5-5 查看 GATHER_STATS_JOB 的当前运行状态。

```
SQL> select job_name,state,owner
  2  from dba_scheduler_jobs;

JOB_NAME                        STATE              OWNER
-------------------------       ---------------    --------------------
PURGE_LOG                       SCHEDULED          SYS
FGR$AUTOPURGE_JOB               DISABLED           SYS
GATHER_STATS_JOB                SCHEDULED          SYS
AUTO_SPACE_ADVISOR_JOB          SCHEDULED          SYS
RLM$EVTCLEANUP                  SCHEDULED          EXFSYS
RLM$SCHDNEGACTION               SCHEDULED          EXFSYS
```

从输出可以看出，该作业的 STATE 为 SCHEDULED，也就是正被“调用”。默认情况下，Oracle Scheduler 维护窗口的周一到周五晚上十点到早上六点以及周末的全天打开。

自动数据收集会按照一定的规则对数据库对象的相关数据进行统计，这些规则如下：

- 不是所有对表的所有行进行统计，Oracle 根据内部算法实现一个采样百分比。
- 不是在维护窗口内对所有的表收集统计数据，没有变化的表只被收集一次。
- 对于新表的第一次统计会统计该表的所有数据行。

下面我们通过数据字典 DBA_TABLES 查询用户 SCOTT 拥有表的统计分析情况。

例子 5-6 查询用户 SCOTT 拥有表的统计分析情况。

```
SQL> select last_analyzed,table_name,owner,num_rows,sample_size
  2  from dba_tables
  3  where owner='SCOTT';

LAST_ANALYZED  TABLE_NAME           OWNER         NUM_ROWS  SAMPLE_SIZE
-------------- -------------------- ------------- --------- --------
28-5月 -10     DEPT                 SCOTT            4          4
29-5月 -10     EMP                  SCOTT           14         14
28-5月 -10     BONUS                SCOTT            0          0
```

```
28-5月 -10     SALGRADE                    SCOTT         5          5
28-5月 -10     EMP_TEST                     SCOTT        14         14
28-5月 -10     NEW_TEST                     SCOTT        14         14

已选择 5 行。
```

输出显示用户 SCOTT 的所有表都被统计过，列 LAST_ANALYZED 表示被统计的时间，其中 NUM_ROWS 说明该表具有的行数，而 SAMPLE_SIZE 说明统计的采样行数，因为是第一次对该表分析，所以对所有行进行统计。

5.5 手工统计数据库数据

除了为 CBO 自动收集统计数据外，还可以使用 DBMS_STATS 包手工收集统计数据，其实自动数据收集的 GATHER_STATS_JOB 作业，本质上也是使用 DBMS_STATS 包来实现数据收集，区别就看是否是 Oracle 自动执行的内部行为。

通过手工收集的数据库对象的统计数据，以及统计的操作系统的统计数据，CBO 计算执行物理查询计划的操作成本。本节我们先介绍如何手工统计数据库数据，在下节会介绍如何统计操作系统的数据。

DBMS_STATS 包提供几个过程来统计不同粒度的数据，粒度分为统计数据库、模式、表以及索引，对应的过程如下例所示。

- GATHER_DATABASE_STATISTICS 为全库中的表统计数据。
- GATHER_SCHEMA_STATISTICS 为某个模式统计数据。
- GATHER_TABLE_STATISTICS 为某个特定的表统计数据。
- GATHER_INDEX_STATISTICS 为某个索引表统计数据。

这统计数据会存储在 DBA_TAB_STATISTICS 和 DBA_TAB_COL_STATISTICS 数据字典中。下面我们通过几个例子说明如何手工统计数据。

1．手工收集特定模式的统计数据

例子 5-7 为模式 SCOTT 的所有表统计数据。

```
SQL> execute dbms_stats.gather_schema_stats(ownname=>'scott');

PL/SQL 过程已成功完成。
```

为了验证是否统计成功，我们使用数据字典 DBA_TABLES 查看模式 SCOTT 所有表的最后分析时间，如下例所示。

例子 5-8 验证模式 SCOTT 的数据统计是否成功。

```
SQL> select last_analyzed,table_name,owner,num_rows,sample_size
  2  from dba_tables
  3  where owner='SCOTT';

LAST_ANALYZED  TABLE_NAME       OWNER      NUM_ROWS  SAMPLE_SIZE
-----------  ------------------------  ---------  ----------  -------
```

```
30-5月 -10    DEPT        SCOTT       4      4
30-5月 -10    EMP         SCOTT      14     14
30-5月 -10    BONUS       SCOTT       0      0
30-5月 -10    SALGRADE    SCOTT       5      5
30-5月 -10    EMP_TEST    SCOTT      14     14
30-5月 -10    NEW_TEST    SCOTT      14     14

已选择 5 行。
```

从输出看到此时的 LAST_ANALYZED 列的值已经是笔者为模式 SCOTT 执行手工统计数据的时间。

2. 手工收集特定表的统计数据

下面介绍统计特定表的过程使用方法，首先看过程 GATHER_TABLE_STATS 的语法结构。

```
DBMS_STATS.GATHER_TABLE_STATS (
   ownname            VARCHAR2,
   tabname            VARCHAR2,
   partname           VARCHAR2 DEFAULT NULL,
   estimate_percent   NUMBER   DEFAULT NULL,
   block_sample       BOOLEAN  DEFAULT FALSE,
   method_opt         VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree             NUMBER   DEFAULT NULL,
   granularity        VARCHAR2 DEFAULT 'DEFAULT',
   cascade            BOOLEAN  DEFAULT FALSE,
   stattab            VARCHAR2 DEFAULT NULL,
   statid             VARCHAR2 DEFAULT NULL,
   statown            VARCHAR2 DEFAULT NULL,
   no_invalidate      BOOLEAN  DEFAULT FALSE);
```

常用的参数为 ownname 和 tabname，如下例收集表 EMP 的统计数据。

例子 5-9 为 SCOTT 用户的表 EMP 统计数据。

```
SQL> execute dbms_stats.gather_table_stats('scott','emp');

PL/SQL 过程已成功完成。
```

3. 手工收集特定索引的统计数据

接下来我们为 SCOTT 模式下某个索引统计数据。此时先查询 SCOTT 模式索引名称，如下例所示。

例子 5-10 查询 SCOTT 模式下的索引。

```
SQL> select index_name,table_name
  2  from dba_indexes
  3  where owner='SCOTT';

INDEX_NAME                    TABLE_NAME
---------------------------------------------------- --------------------
PK_DEPT                        DEPT
PK_EMP                         EMP
```

下面给出过程 GATHER_INDEX_STATS 的使用语法结构。

```
DBMS STATS.GATHER INDEX STATS (
   ownname              VARCHAR2,
   indname              VARCHAR2,
   partname             VARCHAR2 DEFAULT NULL,
   estimate percent     NUMBER   DEFAULT NULL,
   stattab              VARCHAR2 DEFAULT NULL,
   statid               VARCHAR2 DEFAULT NULL,
   statown              VARCHAR2 DEFAULT NULL,
   degree               NUMBER   DEFAULT NULL,
   granularity          VARCHAR2 DEFAULT 'DEFAULT',
   no_invalidate        BOOLEAN  DEFAULT FALSE);
```

我们对表 DEPT 的主键索引 PK_DEPT 进行统计，如下例所示。

例子 5-11 为表 DEPT 的索引统计数据。

```
SQL> execute dbms_stats.gather_index_stats('SCOTT','PK_DEPT');

PL/SQL 过程已成功完成。
```

结果提示过程执行成功，我们通过数据字典 DBA_INDEXES 查看索引的最后统计时间，如下例所示。

例子 5-12 验证主键索引是否成功统计。

```
SQL> select index name,last analyzed
  2  from dba indexes
  3  where owner='SCOTT';

INDEX NAME                    LAST ANALYZED
------------------------- -------------------------------------
PK DEPT                      30-5 月 -10
PK_EMP                       30-5 月 -10
```

数据字典 DBA_INDEXES 中列 LAST_ANALYZED 表示最近一次统计索引的时间。从输出可见，索引 PK_DEPT 和 PK_EMP 已经被统计。

4．手工收集数据库级别的统计数据

在手工收集数据库级别的统计数据之前，需要对初始化参数 JOB_QUEUE_PROCESSES 设置一个非 0 值，才能保证过程 GATHER_DATABAES_STATS 正常工作。否则如果参数 JOB_QUEUE_PROCESSES 为 0，则过程 GATHER_DATABAES_STATS 不会工作。如下例所示，我们查询当前数据库上参数 JOB_QUEUE_PROCESSES 的值。

例子 5-13 查询参数 JOB_QUEUE_PROCESSES 的值。

```
SQL> show parameter job_queue_processes;

NAME                           TYPE         VALUE
------------------------ ----------    --------------------
job_queue_processes           integer       10
```

输出说明当前的 JOB_QUEUE_PROCESSES 参数值为 10。



如果该值为 0，需要使用 ALTER SYSTEM SET 指令修改其值为一正值，并且使用 SCOPE=BOTH 参数，这样使得参数的修改立即生效。如下例所示。

```
SQL> alter system set job_queue_processes=20 scope=both;

系统已更改。
```

下面，我们给出 GATHER_DATABAES_STATS 过程的几个参数的含义，如下所示。

- ESTIMATE_PERCENT：该参数说明统计数据时对数据行的选择百分比。如果该参数为 NULL，说明要对所有表的所有行进行统计计算。
- EMTHOD_OPT：该属性说明是否收集直方图。如果使用 AUTO 则告诉数据库依据列数据的分布以及列的数据负载收集直方图。
- GRANULARITY：该参数是对表而言的。如果该参数为 ALL 说明对所有表的分区、子分区以及全局统计量进行数据统计，该参数可以是 AUTO、GLOBAL AND PARTITION、PARTITION 以及 SUBPARTITION，用于说明统计数据的粒度。
- CASCADE：表示对所有的表以及包含的索引进行数据统计，其实这相当于运行 GATHER_INDEX_STATS 过程，该参数布尔值默认是 FALSE。
- OPTIONS：该参数说明数据库对统计对象做出选择。如果该参数为 GATHER，说明搜集所有对象的统计数据；如果该参数为 GATHER AUTO，只为数据库认为必要的数据库对象统计数据；如果该参数为 GATHER EMPTY，说明只收集目前没有统计数据的数据库对象数据；如果该参数为 GATHER STALE，说明只收集过时的数据库对象数据。

下面我们演示如何手工收集数据库级的统计数据。

例子 5-14 手工收集整个数据库的统计数据。

```
SQL> begin
  2  dbms stats.gather database stats(estimate percent=>null);
  3  end ;
  4  /

PL/SQL 过程已成功完成。
```

上例我们将 ESTIMATE_PERCENT 参数设置为 NULL，说明要为所有的数据行进行统计。

手工统计的数据库数据存储在哪里？Oracle 在数据字典 DBA_TAB_STATISTICS 中存储为 CBO 计算所需的统计数据，在数据字典 DBA_TAB_COL_STATISTICS 收集列的统计数据。如下例所示，我们查询表 EMP 的统计数据。

例子 5-15 查询表 EMP 的统计数据。

```
SQL> select num_rows,avg_space,avg_row_len, num_freelist_blocks,
    last_analyzed
  2  from dba_tab_statistics
  3  where table_name='EMP';
```

```
  NUM_ROWS  AVG_SPACE AVG_ROW_LEN NUM_FREELIST_BLOCKS LAST_ANALYZED
---- --------- ----------- ---------------- -------------------
        14          0         37                  0   29-5月 -10
```

接着，我们继续查看表 EMP 的列的统计数据，如下例所示。

例子 5-16 查看表 EMP 的列的统计数据。

```
SQL> col column_name for a10
SQL>select column_name,num_distinct,low_value,
    high_value,sample_size,avg_col_len
  2  from dba_tab_col_statistics
  3* where table_name='EMP'

COLUMN_NAM NUM_DISTINCT LOW_VALUE  HIGH_VALUE SAMPLE_SIZE AVG_COL_LEN
----- -------------------- -------------- ------------- ----------- ------
EMPNO       14       C24A45       C25023          14          4
ENAME       14       4144414D53   57415244        14          5
JOB         5        414E414C59   53414C4553      14          8
                     5354         4D414E

MGR         5        C24C43       C25003          13          4
HIREDATE    13       77B40C1101   77BB051701      14          8
                     0101         0101

SAL         12       C209         C233            14          4
COMM        4        80           C20F            4           2

COLUMN_NAM NUM_DISTINCT LOW_VALUE  HIGH_VALUE SAMPLE_SIZE AVG_COL_LEN
------- --------------------- ------------- --------- --------- ---------
DEPTNO      3        C10B       C11F              14          3

已选择 8 行。
```

其中，我们查询了该表的具有不同值的列数量、某列中的最大值和最小值、采样数以及列的长度。CBO 在确定一个涉及该表行数据的 SQL 语句时，会考虑为该表统计的行数据而做出最佳执行计划的选择。

5.6 统计操作系统数据

我们已经分析过，当 CBO 选择最佳查询路径时，需要使用数据库对象如表、索引等的统计数据，以及操作系统的统计数据如 I/O 速度、CPU 周期等进行 SQL 的操作耗时计算，选择花费时间最少的执行计划为最佳执行计划。

Oracle 使用 GATHER_SYSTEM_STATS 过程来统计操作系统数据，此时统计的操作系统数据保存在表 SYS.AUX_STATS$中。我们先来看过程 GATHER_SYSTEM_STATS 的语法。

```
DBMS_STATS.GATHER_SYSTEM_STATS(
  gathering_mode     VARCHAR2 DEFAULT 'NOWORKLOAD',
  interval           INTEGER DEFAULT NULL,
```

```
    stattab             VARCHAR2 DEFAULT NULL,
    statid              VARCHAR2 DEFAULT NULL,
    statown             VARCHAR2 DEFAULT NULL);
```

其中参数 gathering_mode 默认值为 NOWORKLOAD，该参数也可以设置为 WORKLOAD，含义如下所示。

- NOWORKLOAD: 无负载模式下运行系统数据统计，此时的数据是安装数据库的计算机自身的常规特性。
- WORKLOAD: 负载模式下运行系统数据统计，此时需要在典型的数据库负载间隔内统计系统数据，这样的数据才能反映数据库的真实系统环境。

而参数 interval 说明在特定的时间间隔内统计数据，如果不使用该参数，也可以使用 START 和 STOP 关键字自己决定何时开始何时结束统计。

通过例子演示如何使用过程 GATHER_SYSTEM_STATS 来收集操作系统数据，如下例所示。

例子 5-17　在无负载方式下收集 10 分钟的系统统计数据。

```
SQL> execute dbms_stats.gather_system_stats('NOWORKLOAD',10);

PL/SQL 过程已成功完成。
```

我们也可以先启动系统数据收集，再适当的时刻停止系统数据收集，然后查询在这段时间内统计的系统数据信息，如下例所示。

例子 5-18　收集系统统计数据。

```
SQL> execute dbms_stats.gather_system_stats('start');

PL/SQL 过程已成功完成。

SQL> execute dbms_stats.gather_system_stats('stop');

PL/SQL 过程已成功完成。
```

上述两个指令间隔 3 分钟执行，然后，通过数据字典 SYS.AUX_STATS$查询这段时间内统计的系统数据。

例子 5-19　查询统计的系统数据。

```
SQL> select *
  2  from sys.aux_stats$;

SNAME                PNAME                PVAL1  PVAL2
---------------- ----------------- ---------- ------------------
SYSSTATS_INFO        STATUS                    COMPLETED
SYSSTATS_INFO        DSTART                    05-30-2010 13:09
SYSSTATS_INFO        DSTOP                     05-30-2010 13:12
SYSSTATS_INFO        FLAGS                     1
SYSSTATS_MAIN        CPUSPEEDNW                537.074
SYSSTATS_MAIN        IOSEEKTIM                 13.493
SYSSTATS_MAIN        IOTFRSPEED                4095
```

```
SYSSTATS_MAIN        SREADTIM                 8.557
SYSSTATS_MAIN        MREADTIM
SYSSTATS_MAIN        CPUSPEED                 491
SYSSTATS_MAIN        MBRC

SNAME                PNAME                   PVAL1  PVAL2
--------------- ---------------- ------- ----------------------
SYSSTATS_MAIN        MAXTHR
SYSSTATS MAIN        SLAVETHR

已选择 13 行。
```

以上输出显示在时间 05-30-2010 13:09 与时间 05-30-2010 13:12 的时间段内统计的系统数据，下面我们解释 PNAME 列参数含义。

- **CPUSPEEDNW**：无负载方式的平均 CPU 周期数。
- **IOSEEKTIM**：寻找时间、等待时间以及操作系统开销之和（毫秒）。
- **IOTFRSPEED**：I/O 传输速度（字节/毫秒）。
- **SREADTIM**：随机读取单个数据块的平均时间。
- **MREADTIM**：顺序读取单个数据块的平均时间。
- **CPUSPEED**：负载方式下的平均 CPU 周期数。
- **MBRC**：一次多块读的平均数据块数。
- **MAXTHR**：最大系统 I/O 吞吐量（字节/秒）。
- **SLAVETHR**：平均 I/吞吐量（字节/秒）。

5.7 手工统计字典数据

除了手工收集系统数据以及数据库统计数据，为了 CBO 优化的目的，也需要收集字典数据，这里分动态性能视图（以$符号开头的数据字典视图）和数据字典（如 DBA_TABLES）。对于动态性能视图的字典数据收集使用 GATHER_FIXED_OBJECTS_STATS 过程，如下例所示。

例子 5-20 收集固定字典表的统计数据。

```
SQL> execute dbms_stats.gather_fixed_objects_stats;

PL/SQL 过程已成功完成。
```

对于数据字典表使用 GATHER_DICTIONARY_STATS 过程来收集，如下例所示。

例子 5-21 收集数据局字典表的统计数据。

```
SQL> execute dbms_stats.gather_dictionary_stats;

PL/SQL 过程已成功完成。
```

也可是使用如下方式实现，即使用过程 GATHER_SCHEMA_STATS 实现。

例子 5-22　使用过程 GATHER_SCHEMA_STATS 统计数据字典数据。

```
SQL> execute dbms stats.gather schema stats('sys');

PL/SQL 过程已成功完成。
```



在执行动态性能视图字典统计数据和数据字典统计数据之前，必须具备 SYSDBA 权限。

5.8　主动优化SQL语句

主动优化 SQL 语句是指在数据库系统的设计、编码阶段，DBA 根据高效 SQL 语句的规则来规范化 SQL 语句的编写，在深入的了解系统需求的基础上（比如哪种查询较多、涉及哪些列属性等）有针对性的优化。下面我们分几节进行讨论。

5.8.1　Where 谓词的注意事项

在 SQL 查询中，通过有效的 WHERE 子句可以明显地减少数据 I/O、减少查询时间。如果对于多表的查询没有使用 WHERE 子句或者无效的 WHERE 子句，则会造成笛卡尔积连接操作，显然这样的操作将成倍地增加查询时间，耗费 CPU、内存以及 I/O 资源。在使用 WHERE 子句时，要注意以下几点：

1．建立基于函数的索引

如果对于一个查询经常用到某个函数，比如使用 SQL 的函数 UPPER，我们有以下的查询。

```
SQL> select *
  2  from scott.emp
  3  where ename='smith';

未选定行
```

显然，这样的查询失败是因为没有 ename='smith'的数据行存在，这里的问题是在 SCOTT 模式的 EMP 表中 ENAME 的值都为大写。如下的查询就会生效。

例子 5-23　通过 UPPER()函数查询用户 smith 的信息。

```
SQL>select *
  2  from scott.emp
  3* where ename=upper(&ename)
输入 ename 的值：  'smith'
原值    3: where ename=upper(&ename)
新值    3: where ename=upper('smith')

    EMPNO ENAME      JOB       MGR HIREDATE    SAL   COMM     DEPTNO
------------ ---------- -------- --------------- ---------- ----- ----
     7359   SMITH      CLERK      7902 17-12 月-80    800                20
```

此时，我们使用一个绑定变量来代替，并且该绑定变量作为 SQL 函数 UPPER 的参数，此时

我们输入小写的 ENAME，即可成功查询，但是注意此时没有使用任何索引，这样的情况下最好使用基于 SQL 函数的索引。如下例所示，创建基于函数的索引。

例子 5-24 创建基于函数 UPPER()的索引。

```
SQL> create index scott emp upper ename on scott.emp(upper(ename));

索引已创建。
```

如果读者再次执行例子 5-23 的查询，使用 EXPLAIN 工具会查询到使用了我们刚刚创建的索引。

在存在多个表的查询时，往往需要表之间的连接操作，而选择合适的连接对查询性能影响很大，所以在选择连接方法时，尽量使用等价连接。使用带 WHERE 子句的等价连接可以有效减少读取的数据量，在使用 WHERE 子句时，最好将具有较高选择性的表作为驱动表，以减少连接的行数。

在使用 WHERE 谓词时，最好将 EXITS 替换为 IN，这样可以更好的执行子查询。而在使用 HAVING 子句的地方，最好使用 WHERE 子句替换，因为 WHERE 子句较早地限制了查询的数据行数。

5.8.2 SQL 语句优化工具

SQL 语句执行计划就是解释 SQL 语句的执行步骤。在 Oracle 中使用 explain plan for 指令来获得 SQL 语句的执行计划，也可以使用 autotrace 执行获得 SQL 语句的执行过程和相关的统计信息，如物理读的数据量、磁盘内排序的数据量等。下面我们分别介绍这两种常用的、获得 SQL 语句执行计划的指令。

1. 使用 EXPLAIN PLAN FOR 命令

在使用该指令时，必须先使用一个脚本文件 utlxplan.sql 来创建 plan_table 以存储使用 explain plan for 语句获得的 SQL 语句的分析结果。该脚本文件保存在$ORACLE_HOME\ RDBMS\ADMIN 目录下。

其实 utlxplan.sql 就是创建表 PLAN_TABLE 的脚本程序，我们打开该文件可以清楚地看到创建表的说明和建表的 SQL 语句。如下例所示，我们给出脚本 utlxplan.sql 的部分内容。

```
Rem This is the format for the table that is used by the EXPLAIN PLAN
Rem statement.  The explain statement requires the presence of this
Rem table in order to store the descriptions of the row sources.

create table PLAN_TABLE (
        statement_id        varchar2(30),
        plan_id             number,
        timestamp           date,
        remarks             varchar2(4000),
        operation           varchar2(30),
        options             varchar2(285),
        ………省略了部分列属性的定义
        other_xml           clob
);
```

该脚本文件的说明部分清楚地介绍创建该表的目的就是存储对于执行计划的过程描述信息。这里就是告诉读者，在 Oracle 数据库中有很多类似的 SQL 脚本文件，只要打开这些文件分析一下，一切都明晰了，并且读者也可以模仿写出规范的 SQL 脚本文件。

下面，我们执行该脚本文件，如下例所示。

例子 5-25　执行脚本文件 utlxplan.sql。

```
SQL> @F:\oracle\product\10.2.0\db_1\RDBMS\ADMIN\utlxplan.sql

表已创建。
```

显然，表已经创建。在打开的 utlxplan.sql 脚本文件中，可以看到表名为 PLAN_TABLE。我们查询该表的结构，同时也是为了验证该表是否存在。

例子 5-26　查看表 PLAN_TABLE 的结构。

```
SQL> desc plan_table;
 名称                                          是否为空? 类型
 ------------------------------- --------- -----------------
 STATEMENT_ID                                  VARCHAR2(30)
 PLAN ID                                       NUMBER
 TIMESTAMP                                     DATE
 REMARKS                                       VARCHAR2(4000)
 OPERATION                                     VARCHAR2(30)
 OPTIONS                                       VARCHAR2(285)
 OBJECT NODE                                   VARCHAR2(128)
 OBJECT OWNER                                  VARCHAR2(30)
………..
TIME                                           NUMBER(38)
 QBLOCK NAME                                   VARCHAR2(30)
 OTHER_XML                                          CLOB
```

现在我们创建了为了执行 explain plan for 指令所需的表，接下来就可以使用该指令执行 SQL 语句的分析了。我们通过下例说明如何使用 explain plan for 指令分析 SQL 语句执行计划。

例子 5-27　通过 explain plan for 指令分析 SQL 语句的执行计划。

```
SQL> explain plan for
  2  select count(*) from scott.emp;

已解释。
```

此时解释了 SQL 语句 select count(*) from scott.emp，该语句的作用是计算用户 SCOTT 的表中记录的行数。其执行分析信息存储在表 PLAN_TABLE 中，我们查询该语句的执行计划，如下例所示。

例子 5-28　查看表 PLAN_TABLE 中的 SQL 语句执行计划信息。

```
SQL> col id for 999
SQL> col operation for a20
SQL> col options for a20
SQL> col object_name for a20
```

```
SQL> select id,operation,options,object_name,position
  2  from plan_table;

 ID OPERATION              OPTIONS          OBJECT_NAME   POSITION
------ ------------------------ ---------------- ------------------
   0  SELECT               STATEMENT                           3
   1  SORT                  AGGREGATE                          1
   2  TABLE ACCESS          FULL              EMP              1
```

从上例的输出中可以看出 SQL 语句的执行过程。我们分析最后一行，ID 说明步骤标识，OPERATION 为 TABLE ACCESS 说明该步骤的行为是访问表，OPTIONS 为 FULL 说明，使用全表扫描访问表，OBJECT_NAME 说明行为的对象为表 EMP。

在分析一个 SQL 语句时，经常使用该指令判断是否使用适当的索引完成表的访问，从而适当的修改索引或创建索引来优化 SQL 语句的执行。下面我们继续分析如何使用 AUTOTRACE 指令分析 SQL 语句的执行计划。

1. 使用 AUTOTRACE 命令

使用 AUTOTRACE 指令可以跟踪 SQL 语句、分析其执行步骤、统计相关信息如物理读数据量、磁盘和内存排序数据量等。但是要执行该指令需要设置如下几个参数。

- SQL_TRACE：该参数说明是否启动对 SQL 语句的追踪。默认该参数为 FALSE，要启用 AUTOTRACE 功能需要将参数 SQL_TRACE 设置为 TRUE，该参数可以动态改变。注意，在不需要追踪 SQL 语句时，最好将该参数设置为 FALSE，因为它会造成跟踪所有执行的 SQL 语句，这样会产生大量的 TRC 文件，对磁盘空间有一定的冲击。
- USER_DUMP_DEST：该参数说明 SQL 语句追踪文件的记录位置，在笔者的计算机上其默认目录为 F:\oracle\product\10.2.0\admin\orcl\udump。
- TIMED_STATISTICS：该参数可以使用 ALTER SYSTEM 或 ALTER SESSION 动态设置。默认参数值为 TRUE。

所以，我们只需要设置参数 SQL_TRACE 来启动对 SQL 语句执行的追踪，如下例所示。

例子 5-29 设置参数 SQL_TRACE 启动 SQL 语句追踪。

```
SQL> alter system set sql trace = true;

系统已更改。
```

下面，我们查询当前系统上是否启用 SQL 语句追踪功能，如下例所示。

例子 5-30 查询 SQL_TRACE 参数值。

```
SQL> show parameter sql trace;

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------
sql_trace                            boolean     TRUE
```

现在，我们已经做好了使用 AUTOTRACE 指令查看 SQL 语句执行计划的准备。下面我们给出一个例子说明如何使用该指令，并解释相关参数。

例子 5-31 使用 AUTOTRACE 追踪 SQL 语句执行计划。

```
SQL> set autotrace traceonly;
SQL> select count(*) from scott.emp;


执行计划
----------------------------------------------------------------------
Plan hash value: 2083855914

----------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Cost  (%CPU)| Time     |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE     |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| EMP  |    55 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------

统计信息
----------------------------------------------------------------------
       224  recursive calls
         0  db block gets
        28  consistent gets
         5  physical reads
         0  redo size
       408  bytes sent via SQL*Net to client
       385  bytes received via SQL*Net from client
         2  SQL*Net roundtrips to/from client
         4  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

上例中，启动 AUTOTRACE 并仅执行 TRACE 功能，其实还有解释 EXPLAIN 和统计 STATISTICS 功能。AUTOTRACE 结果分两部分，一部分是 SQL 语句的执行计划，一部分是统计信息。

从执行计划可以清楚看出 SQL 语句的执行步骤、访问的对象以及消耗的 CPU，如果是表，还记录了访问的表的行数。

统计信息显示了更具体的数据访问和磁盘访问的细节，如物理读数据量、重做数据量、迭代访问传送到客户端的数据量、客户端传递给数据库服务器的数据量，内存排序的数据量，以及磁盘排序的数据量。其实如果此处出现磁盘排序，即 sorts(disk)的值不为 0，或许需要调整 PGA 中 SORT_AREA_SIZE 的大小（排序区自动管理的情况下），或调整 PGA_AGGREGATE_TARGET 参数以增减 PGA 的大小，使得排序行为尽可能在内存完成。

下面我们详细介绍统计信息中每一行的含义。

- recursive calls: 递归调用的次数。
- db block gets : 读数据块的数量。
- consistent gets: 总的逻辑 I/O。
- physical reads: 物理 I/O。

- redo size：重做数量。
- bytes sent via SQL*Net to client：SQL*Net 通信。
- bytes received via SQL*Net from client。
- SQL*Net roundtrips to/from client。
- sorts (memory)：内存排序统计。
- sorts (disk)：磁盘排序统计。
- rows processed：被检索的行数。

在不需要使用 AUTOTRACE 时，将该功能关闭，否则所有接下来执行的 SQL 语句都会被追踪分析。关闭 AUTOTRACE 功能的指令，如下例所示。

例子 5-32 关闭 AUTOTRACE。

```
SQL> set autotrace off;
```

现在我们看看 AUTOTRACE 的其他功能，如下例所示查看启动 AUTOTRACE 的一些参数。

例子 5-33 查看启动 AUTOTRACE 的其他参数。

```
SQL> set autotrace
用法: SET AUTOT[RACE] {OFF | ON | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
```

如果需要启动 AUTOTRACE 功能而只需要执行计划，使用 SET AUTOTRACE ON explain 指令。如果启动 AUTOTRACE 功能而只需要统计信息，使用 SET AUTOTRACE ON statistics 指令。如果读需要则使用 SET AUTOTRACE TRACEONLY 指令。使用 SET AUTOTRACE ON 指令默认与使用 SET AUTOTRACE TRACEONLY 指令相同，都是启动追踪执行计划和追踪统计信息。

2. TKPROF

TKPROF 是分析 SQL Trace 文件的实用程序，它将 SQL Trace 文件格式化，得到一个 SQL 语句的执行结果。但是，其前提是必须启动 SQL Trace。

下面我们首先介绍如何启动 SQL Trace。

（1）启动 SQL Trace 的前提

要启动 SQL Trace 必须设置初始化参数使得数据库可以在不同级别上追踪 SQL 语句。我们需要理解五个参数的含义，并做相应的设置。五个参数的含义如下所示。

- STATISTICS_LEVEL：该参数有三个值即 TYPICAL、ALL 和 BASIC。如果该参数值为 TYPICAL、ALL 则自动统计 SQL 语句。如果该参数值为 BASIC，则需要将另一个参数 TIMED_STATISTICS 设置为 TRUE 启动数据库自动收集 SQL 数据。
- TIMED_STATISTICS：在参数 STTISTICS_LEVEL 值为 BASIC，则需要将参数 TIMED_STATISTICS 设置为 TRUE 启动数据库自动收集 SQL 数据。如果参数
- STATISTICS_LEVEL 的值为 TYPICAL 或 ALL，必须将参数 IMED_STATISTICS 设置为 FALSE 来关闭 SQL 语句执行数据的自动收集。
- USER_DUMP_DEST：该参数存储 SQL 语句的追踪文件。通过下例查询当前数据库的 SQLTrace 文件目录，如下例所示。

例子 5-34　查询当前数据库的 SQLTrace 文件目录。

```
SQL> show parameter user_dump_dest;

NAME                    TYPE           VALUE
----------------------- -------------------------------------
user_dump_dest     string          E:\ORACLE\PRODUCT\10.2.0\ADMIN
                              \LSZPRI\UDUMP
```

- MAX_DUMP_FILE_SIZE: 设置 USER_DUMP_DEST 参数指定的目录存储 SQL 追踪文件的大小，默认该参数值为 UNLIMITED，如下查询所示。

例子 5-35　查询当前数据库参数 MAX_DUMP_FILE_SIZE 的值。

```
SQL> show parameter max_dump_file_size;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
max_dump_file_size                   string       UNLIMITED
```

（2）启动 SQLTrace 追踪

可以在实例级以及会话级启动 SQL 追踪。在实例级启动 SQLTrace 追踪使用 ALTER SYSTEM 指令，在会话级启动 SQLTrace 追踪使用 ALTER SESSION 指令或使用 Oracle 提供的包 DBMS_SESSION 来启动。下面通过例子分别说明。

例子 5-36　在实例级启动 SQLTrace 追踪。

```
SQL> alter system set sql_trace=true scope=both;

系统已更改。
```

一旦在整个实例级上启动 SQL 追踪，会出现大量的 SQL 文件，所以考虑对数据库系统的性能影响最好关闭实例级的 SQL 追踪，而是在必要的会话中启动 SQL 追踪，如下例所示，在当前会话中启动 SQL 追踪。

例子 5-37　在会话级启动 SQL 追踪。

```
SQL> alter session set sql_trace=true;

会话已更改。
```

也可以使用 DBMS_SESSION 工具包启动会话级的 SQL 追踪，如下例所示。

例子 5-38　在会话级启动 SQL 追踪。

```
SQL> begin
  2  sys.dbms_session.set_sql_trace(true);
  3  end;
  4  /

PL/SQL 过程已成功完成。
```

（3）使用 TKPROF 解释 SQLTrace 文件

在会话级设置了 SQL 追踪之后，我们就可以通过 TKPROF 工具来分析 SQL 语句的真实执行结果。因为原始的 SQL 追踪文件内容很多，既没有归类也不易阅读，所以通过 TKPROF 工具格式化后，就很容易分析。

我们先看 TKPROF 的用法，如下例所示。

例子 5-39 TKPROF 的用法。

```
    C:\Documents and Settings\Administrator>tkprof
    Usage: tkprof tracefile outputfile [explain= ] [table= ]
                [print= ] [insert= ] [sys= ] [sort= ]
      table=schema.tablename   Use 'schema.tablename' with 'explain=' option.
      explain=user/password     Connect to ORACLE and issue EXPLAIN PLAN.
      print=integer          List only the first 'integer' SQL statements.
      aggregate=yes|no
      insert=filename        List SQL statements and data inside INSERT statements.
      sys=no              TKPROF does not list SQL statements run as user SYS.
      record=filename        Record non-recursive statements found in the trace
file.
      waits=yes|no      Record summary for any wait events found in the trace file.
      sort=option       Set of zero or more of the following sort options:
```

下面我们执行一个 SQL 查询之后，通过 TKPFOR 分析该 SQL 语句的执行结果。

例子 5-40 执行 SQL 查询。

```
    SQL> select ename,sal,mgr
      2  from scott.emp
      3  where sal>2000;

    ENAME              SAL       MGR
    -------------------------- ---------- ----------
    JONES               2975       7839
    BLAKE               2850       7839
    CLARK               2450       7839
    SCOTT               3000       7555
    KING                5000
    FORD                3000       7555

    已选择 5 行。
```

在 USER_DUMP_DEST 参数指定的目录下会产生追踪文件 lszpri_ora_2375.trc。下面我们通过 TKPROF 工具来分析 SQL 语句的执行，如下例所示。

例子 5-41 使用 TKPROF 工具格式化 SQL 追踪文件。

```
    E:\oracle\product\10.2.0\admin\lszpri\udump>tkprof      lszpri_ora_2375.trc
mytrace.txt sys=no

    TKPROF: Release 10.2.0.1.0 - Production on 星期二 5 月 1 17:42:24 2010

    Copyright (c) 1982, 2005, Oracle.  All rights reserved.
```

在上述格式化过程中，将格式化后的数据保存在当前目录的文件 mytrace.txt 中，并且不分析和 SYS 用户相关的 SQL 语句。下面我们分析文件 mytrace.txt 中的内容，这是我们学习的关键。因为文件内容很多，我们分割文件内容分别解释。

```
TKPROF: Release 10.2.0.1.0 - Production on 星期二 5月 1 17:48:54 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Trace file: lszpri_ora_2375.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************
```

上面这段信息是 TKPROF 格式化后的文件开头部分，说明对于每一个 SQL 语句都分析如上所示的内容。以下是部分输出内容。

```
OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
---- ------  ---- ---------- ---------- ---------- ----------  ----------
Parse       25      0.03       0.01          0          0          0           0
Execute    204       0.14       0.27          0          0          0           0
Fetch      231      0.00       0.18         33        530        109         791
---- ------  --------------- ---------- ---------- ----------  ----------
total      450      0.17       0.48         33        530        109         791

Misses in library cache during parse: 20
Misses in library cache during execute: 20

    3  user  SQL statements in session.
  204  internal SQL statements in session.
  207  SQL statements in session.
```

这一部分总结了 SQL 语句执行在每个阶段所耗费的资源，以及是否发生硬解析。下面解释格式化参数的含义。

- CALL：该参数说明 SQL 语句的不同执行阶段。
- COUNT：不同执行阶段所读取的数据块数量。
- CPU：不同执行阶段所消耗的 CPU 时间，单位是秒。
- ELAPSED：执行用掉的时间。
- DISK：物理磁盘数据读操作数目。
- QUERY：一致的缓冲区读取数量。

- CURRENT：数据库块读取的数量。

在上面这段输出中 SQL 语句被分析了 25 次，总共消耗了 0.01 秒的时间，语句被执行了 204 次，执行时间总共花费了 0.27 秒，在解析和执行其间没有磁盘 I/O 和缓冲区读取操作，FETCH 操作被执行了 450 次，共消耗 0.18 秒，涉及了 33 次磁盘读取以及 530 缓冲区读取操作，总共读取了 109 个数据库块，涉及 791 行数据。

在库缓存中丢失的命中次数为 20 次，说明有 20 次的硬解析出现，因为在库缓冲区中没有发现同样的 SQL 语句执行计划。

最后给出一个用户和 SQL 语句的总结，涉及三个用户 SQL 语句，204 个内部 SQL 语句总共涉及 207 个 SQL 语句。在文件的最后给出一个总结，说明 Trace 文件名、文件涉及的会话数，以及 SQL 语句的数量等，总结的内容如下所示。

```
********************************************************************************
Trace file: lszpri_ora_2375.trc
Trace file compatibility: 10.01.00
Sort options: default

       1  session in tracefile.
       3  user  SQL statements in trace file.
     204  internal SQL statements in trace file.
     207  SQL statements in trace file.
      23  unique SQL statements in trace file.
    1779  lines in trace file.
      18  elapsed seconds in trace file.
```

SQLTrace 工具提供了调整 SQL 的格式化输出信息，对 SQL 语句的执行阶段进行了详细的分析，给出对资源的实际消耗情况，并对硬解析给出明确的说明，使得对 SQL 优化更加透明。

5.8.3 使用索引

对于用户经常使用的查询，尤其是大表的查询，在程序设计时应该对这些问题做出预测，并要求对这样的查询尽量使用索引。这样通常可以加快查询速度，减少系统 I/O，但是并不是所有的系统都经过精细的需求分析，如果遇到这样的情况，发现某个 SQL 语句总是使用全表扫描实现用户的查询，则需要通过建立索引，加快查询速度，这是 DBA 可以介入的优化 SQL 语句的方法。我们使用 EXPLAIN PLAN FOR 分析语句的执行中的全表扫描行为。首先把表 PLAN_TABLE 截断，如下例所示。

例子 5-42 截断 PLAN_TABLE 表。

```
SQL> truncate table plan_table;

表被截断。
```

在使用 EXPLAIN PLAN FOR 语句后，表 PLAN_TABLE 中已有数据，这里截断表清空数据，在查询时减少数据量的输出。接下来我们分析一个 SQL 查询，查询 HR 用户中员工的部分信息，并使用 EXPLAIN 分析是否使用了索引。首先使用 HR 用户登录数据库。

例子 5-43　登录数据库。

```
SQL> connect hr/oracle@orcl
ERROR:
ORA-28002: the password will expire within 10 days


已连接。
```



上述输出中的错误提示密码在 10 天内过期，因为笔者启用了默认参数文件，读者可以参考本书相关内容，此时读者完全可以忽略这个问题，继续下面的步骤。

下面使用 SQL 语句查询表 EMPLOYEES 中的 JOB 的种类，如下例所示，我们使用 EXPLAIN PLAN FOR 分析该语句的执行。

例子 5-44　使用 EXPLAIN 分析 SQL 语句的执行。

```
SQL> explain plan for
  2  select job id,count(*)
  3  from employees
  4  group by job_id;

已解释。
```

接下来查询是否使用了分析结果，因为我们没有在 JOB_ID 上建立索引，所以语句的执行结果应该是使用全表扫描，查询结果如下例所示。

例子 5-45　查询 EXPLAIN 的分析结果。

```
SQL>select id,operation,options,object_name,position
  2* from plan_table

ID OPERATION          OPTIONS        OBJECT_NAME     POSITION
--- ------------------ -------------- ---------------- -----------------
  0 SELECT STATEMENT                                      4
  1 HASH               GROUP BY                           1
  2 TABLE ACCESS        FULL          EMPLOYEES            1
```

从上述输出的第三行可以看出表访问使用全表扫描，因为 OPTIONS 值为 FULL，而 OBJECT_NAME 为 EMPLOYEES。下面创建基于列 JOB_ID 的索引。

例子 5-46　创建基于列 JOB 的索引。

```
SQL> create index emp_job_idx
  2  on employees(job_id);

索引已创建。
```

我们使用数据字典 USER_INDEXES 查询该索引的创建信息，如下例所示。

例子 5-47　插叙索引 emp_job_idx 的创建信息。

```
SQL> col index_name for a20
```

```
SQL> col index_type for a15
SQL> col table_owner for a20
SQL> col table_name for a20
SQL> run
  1   select index_name,index_type,table_owner,table_name
  2  from user_indexes
  3* where index_name like 'EMP_JOB%'

INDEX_NAME      INDEX_TYPE      TABLE_OWNER       TABLE_NAME
------------------- --------------------- ----------------- ---------
EMP_JOB_IDX       NORMAL           HR                EMPLOYEES
```

显然索引 EMP_JOB_IDX 是基于列的索引。现在我们可以继续使用 EXPLAIN 执行对 SQL 语句的分析了。先截断表 PLAN_TABLE，该用户必须在 SYSTEM 用户下截断，如下例所示。

例子 5-48 截断表 PLAN_TABLE

```
SQL> connect system/oracle@orcl
已连接。
SQL> truncate table plan_Table;

表被截断。
```

接下来，我们再次解释 SQL 语句的执行，如下例所示。

例子 5-49 使用 EXPLAIN 解析 SQL 语句。

```
SQL> connect hr/oracle@orcl
ERROR:
ORA-28002: the password will expire within 10 days


已连接。
SQL> explain plan for
  2  select job_id,count(*)
  3  from employees
  4  group by job_id;

已解释。
```

在建立了索引后，我们再次查询表 PLAN_TABLE 中记录的分析信息，查看是否使用了索引 EMP_JOB_IDX，如下例所示。

例子 5-50 在创建索引后继续查询 SQL 语句的执行计划。

```
SQL> select id,operation,options,object_name,position
  2  from plan_table;

ID OPERATION              OPTIONS         OBJECT_NAME      POSITION
-- ---------------------- ---------- -------------- -----------
0 SELECT STATEMENT                                              1
1 SORT                  GROUP BY NOSORT                        1
2 INDEX                 FULL SCAN          EMP_JOB_IDX         1
```

显然，从输入的第 3 行可以看出，该表使用了对表 EMPLOYEES 创建的索引 EMP_JOB_IDX。因为在 OPERATION 操作使用了索引 INDEX，而且对 OBJECT_NAME 值为 EMP_JOB_IDX 的索引进行了扫描操作（FULL SCAN）。

在数据库的分析以及设计中，如果需要复杂算法实现的查询，则最好创建基于该算法的函数，然后基于该函数对特定的列创建索引，这在一定程度上会提高查询速度。下面我们以 SCOTT 用户的 EMP 表为例子，说明创建基于函数的索引。我们通过一个简单公式创建基于函数的索引，如下例所示。

例子 5-51 创建基于函数的索引。

```
SQL> create index scott_emp_income_idx
  2  on scott.emp(sal*12);

索引已创建。
```

在上例中，我们在 SCOTT 用户的表 EMP 上基于简单的计算公式创建了索引，索引名为 SCOTT_EMP_INCOME_IDX。在查询员工的年收入相关的查询时，可以使用该索引减少查询时间。

我们使用数据字典 USER_INDEXES 查看是否成功创建索引，如下例所示。

例子 5-52 查看是否成功创建索引 SCOTT_EMP_INCOME_IDX。

```
SQL> col index_name for a20
SQL> col table_owner for a15
SQL> col table_name for a15
SQL> select index_name,index_type,table_owner,table_name
  2  from user indexes
  3* where index_name like 'SCOTT%'

INDEX NAME           INDEX TYPE       TABLE OWNER    TABLE NAME
----------------------------- -------------------- --------------------
SCOTT_EMP_INCOME_IDX FUNCTION-BASED NORMAL  SCOTT          EMP
```

从输出可以看出索引 SCOTT_EMP_INCOME_IDX 记录在数据字典 USER_INDEX 中，属于用户 SCOTT 且基于表 EMP。

我们创建该索引的目的就是在查询时，希望 Oracle 使用索引查询提高响应速度。下面我们使用 EXPLAIN PLAN FOR 解析 SQL 语句，该 SQL 语句包含一个查询，查询表 EMP 中年薪少于 30000 的员工信息。

在使用 EXPLAIN PLAN FOR 之前，先截断表 PLAN_TABLE，如下例所示。

例子 5-53 截断表 PLAN_TABLE。

```
SQL> truncate table plan table;

表被截断。
```

然后，我们执行 EXPLAIN PLAN FOR 指令解析一个 SQL 语句。

例子 5-54 使用 EXPLAIN PLAN FOR 解析 SQL 语句。

```
SQL> explain plan for
```

```
  2  select ename,job,mgr,deptno
  3  from scott.emp
  4  where sal*12<30000;

已解释。
```

上述分析的 SQL 语句中，查询表 EMP 中员工年薪少于 30000 的信息，该语句的 WHERE 子句中包含条件 sal*12<30000，所以此时 Oracle 会使用基于函数的索引。下面我们查看表 PLAN_TABLE 看上述 SQL 语句是否成功使用索引实现查询。

例子 5-55 查询 PLAN_TABLE。

```
SQL> select id,operation,options,object_name
  2  from plan_table;

ID OPERATION                  OPTIONS              OBJECT_NAME
--- ------------------------- -----------------  -----------------
0 SELECT STATEMENT
1 TABLE ACCESS              BY INDEX ROWID       EMP
2 INDEX                     RANGE SCAN          SCOTT_EMP_INCOME_IDX
```

从查询结果的第 2 行(ID 为 1)看出，该查询使用了索引，通过索引中的行 ID（ROWID）查询需要的数据行，操作的对象为 EMP 表。



在使用 EXPLAIN PLAN FOR 指令时，SQL 语句并不真正的执行，而只是分析 SQL 语句的执行。

在 Oracle 中也可以使用动态数据字典 v$sql_plan 查询一个 SQL 语句的执行计划，但是要求必须实际执行了这个 SQL 语句。如下例所示，我们先执行上例中分析的 SQL 语句查询。

例子 5-56 执行对 EMP 表的查询语句。

```
SQL> select ename,job,mgr,deptno
  2  from scott.emp
  3  where sal*12<30000;

ENAME       JOB              MGR      DEPTNO
--------- ------------- ----------- ----------
SMITH       CLERK            7902        20
JAMES       CLERK            7598        30
WARD        SALESMAN         7598        30
MARTIN      SALESMAN         7598        30
MILLER      CLERK            7782        10
TURNER      SALESMAN         7598        30
ALLEN       SALESMAN         7598        30
CLARK       MANAGER          7839        10

已选择 8 行。
```

接着，为了判断上述查询是否使用了索引 SCOTT_EMP_INCOME_IDX，我们使用 v$sql_plan 来查询执行计划，如下例所示。

例子 5-57 使用 v$sql_plan 来查询执行计划。

```
SQL> select id,operation,options,object_name,object_owner
  2  from v$sql_plan
  3 where object_name like 'EMP%';

ID OPERATION       OPTIONS          OBJECT_NAME     OBJECT_OWNER
------ ------------- ---------------- ---------------- ----------------
1 TABLE ACCESS     BY INDEX ROWID    EMP              SCOTT
```

结果显示我们上例的查询 SQL 语句使用了索引，因为 OPTIONS 值为 BY INDEX ROWID。

在 Oracle 10g 之前的版本中，并不是默认使用基于函数的索引，所以需要设置参数 query_rewrite_enabled 为 true。而在 Oracle 10g 中，该参数默认值为 true，即在 Oralce 10g 中默认可以使用基于函数的索引，如下例所示。

例子 5-58 查询参数 query_rewrite_enabled 的值。

```
C:\Documents and Settings\Administrator>sqlplus /nolog

SQL*Plus: Release 10.2.0.1.0 - Production on 星期一 9月 28 20:31:45 2009

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> connect system/oracle@orcl
已连接。
SQL> show parameter query;

NAME                                        TYPE        VALUE
------------------------------------------- ----------- -------------
query_rewrite_enabled                       string      TRUE
query_rewrite_integrity                     string      enforced
```

上例是在 Oracle 10g 版本的数据库服务器上查询参数 query_rewrite_enabled 的值，显然该参数值为 TRUE，说明默认可以使用基于函数的索引。

下面为了更好的理解 AUTOTRACE 的使用，我们再使用 AUTOTRACE 来分析上述查询。

例子 5-59 使用 AUTOTRACE 分析 SQL 语句。

```
SQL> set autotrace traceonly
SQL> select ename,job,mgr,deptno
  2  from scott.emp
  3  where sal*12<30000;

已选择 8 行。


执行计划
--------------------------------------------------------------------------
Plan hash value: 455855571

--------------------------------------------------------------------------
| Id  | Operation     | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
```

```
---------------------------------------------------------------------------
|   0  | SELECT STATEMENT      |        |     8 |   415 |     2      (0)|  00:00:01 |
|   1  |  TABLE ACCESS BY INDEX ROWID| EMP   | 8 |   415 |   2  (0)|  00:00:01 |
|*  2  | INDEX RANGE SCAN | SCOTT_EMP_INCOME_IDX | 1 |  | 1  (0)|  00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------------------------------

   2 - access("SAL"*12<30000)

Note
-----
   - dynamic sampling used for this statement


统计信息
---------------------------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        752  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          8  rows processed
```

读者注意在上例的输出中加粗字体的内容，在执行计划的第二步中 OPERATION 为 **TABLE ACCESS BY INDEX ROWID** 说明该表访问使用索引，第三步扫描索引，通过索引值中那些 sal*12<30000 条件的记录找到相应的行 ID，通过 ROWID 找到所需要访问的行，这样不需要全表扫描表 EMP，而只需要通过索引，通过行 ID 直接检索表 EMP 中满足 sal*12<30000 条件的行。

统计信息中的最后一行“8　rows processed”，说明了此次查询检索的表的行数。最后关闭自动追踪，如下例所示。

例子 5-60 关闭 AUTOTRACE 工具。

```
SQL> set autotrace off;
```

索引是一个很重要的概念，合理的使用索引可以加快数据的搜索时间，但是索引不是越多越好，因为索引本身也需要存储和搜索，对于频繁的更新操作的表，索引的使用会降低数据更新的性能，因为此时索引也需要更新，并且是否使用索引是由 Oracle 决定的。

下面我们总结使用索引的时机。

索引并不是越多越好，使用索引需要对数据库系统的应用环境，以及对用户应用有清晰的了解。我们知道对于 OLTP 系统，有大量的插入或删除操作，对于这样的系统要尽量少的使用索引（只在必要的列上使用索引，如在主键上建索引），因为索引在大量 DML 操作时，会触发自身的更新维护，并且需要存储空间。而对于数据仓库系统，都是用户的查询业务，所以此时尽量多的使用索

引，只会增加系统查询的性能，而不用考虑对索引的维护费用。

如果确定查询一个表的查询结果不会高于总行数的 15%，则可以使用索引加快查询速度；如果超过了 15%，则使用全表扫描往往更有效。如果确定查询一个表的查询结果高于总行数的 15%，且使用了索引，此时索引的使用会阻止全表扫描，对查询性能产生不利的影响。

在以下条件下，一般要考虑使用索引。

- 对 WHERE 子句涉及的列创建索引。
- 对表的外键创建索引。
- 对高选择性的列创建索引，这里高选择性的含义是该列具有相同的值很少。

5.8.4　索引类型及使用时机

1．B 树索引

B 树索引是 Oracle 默认的索引类型，研究 B 树索引也可以帮助理解位图索引和反向键索引，所以本节花较多篇幅讲解 B 树索引。

叶子节点包含索引的实际值和该索引条目的行 ID 即 ROWID。B 树索引的结构如图 5-2 所示。

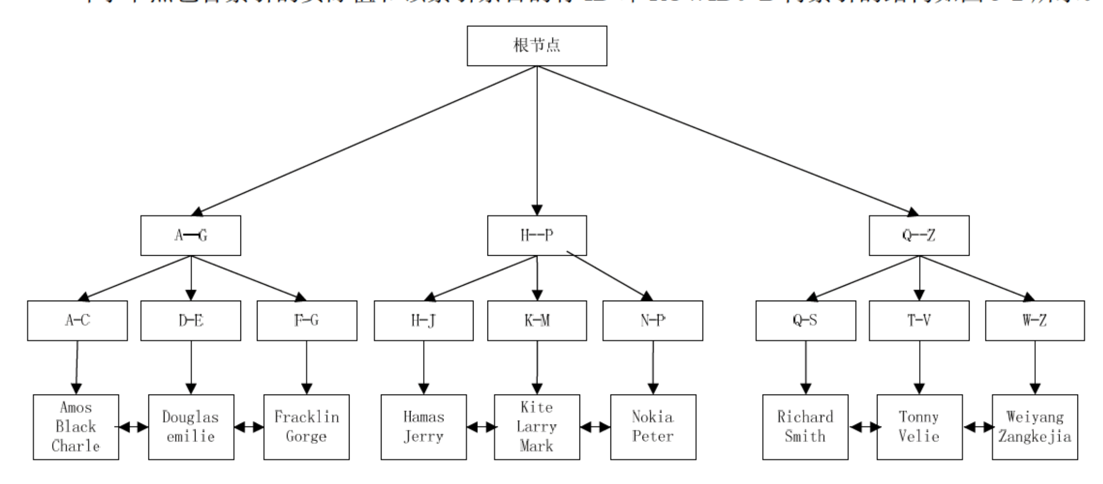


图 5-2　B 树索引结构图

在图 5-2 中，B 树索引结构有三个基本组成部分：根节点、分支节点和叶子节点，其中根节点位于索引结构的最顶端，而叶子节点位于索引结构的最底端，中间为分支节点。

在叶子节点中存储了实际的索引列的值和该列所对应的记录的行 ID，即 ROWID，ROWID 是唯一的 Oracle 指针，指向该行的物理位置，使用 ROWID 是 Oracle 数据库中访问行最快的方法。叶子节点其实是一个双向链表，每个叶子节点包含一个指向下一个和上一个叶子节点的指针，这样在一定范围内便利索引以搜索需要的记录。

每个分支节点又包含其他分支节点，Oracle 设计的 B 树索引结构保证了 B 树索引从根到叶子都有相等的分支节点，保证了 B 树索引的平衡，这样就不会因为基表的数据插入后删除操作造成 B 树索引变得不平衡，从而影响索引的性能。而且如果一个叶子节点为空，则 Oracle 会释放该空间用于它处。

下面给出一个使用 B 树索引的搜索过程，如要查找 Larry，则在根节点转向中间的分支节点，然后继续搜索其下的分支节点，发现需要继续转到它的中间那个子分支节点即 K-M 节点，然后在叶子节点中找到所需要的列的值及其该列所对应的行 ID，从而找到更多的需要的数据。



Oracle 创建的普通索引如果没有说明类型就是 B 树索引。

2. 位图索引

位图索引是 Oracle 10g Enterprise Edition 支持的索引机制。位图索引使用位图标识被索引的列值，它适用于没有大量更新任务的数据仓库。因为使用位图索引时，每个位图索引项与表中大量的行有关联，当表中有大量数据更新、删除和插入时，位图索引相应地需要做大量更改，而且索引所占用的磁盘空间也会明显增加，并且索引在更新时受影响的索引需要锁定，所以位图索引不适合于有大量更新操作的 OLTP 系统。虽然可以通过重建索引类位图位图索引，但是对于有大量更新操作的表，最好不选择使用位图索引。

下面我们以一个 SQL 查询作为例子，解释位图索引的过程，该语句为：

```
SELECT EMPNO,ENAME,SAL
FROM EMP
WHERE JOB = 'SALESMAN';
```

上述查询语句的目的是在 EMP 表中查询工作岗位是 SALESMAN 的员工的员工号、姓名和薪水。此时假设已经在 EMP 表的 JOB 列建立了位图索引，其结构如图 5-3 所示。

Index on JOB

JOB =' CLERK'	1 0 0 0 0 0 0 0 0 0 1 1 0 1
JOB =' SALESMAN'	0 1 1 0 1 0 0 0 0 1 0 0 0 0
JOB =' PRESIDENT'	0 0 0 0 0 0 0 0 1 0 0 0 0 0
JOB =' MANAGER'	0 0 0 1 0 1 1 0 0 0 0 0 0 0
JOB =' ANALYST'	0 0 0 0 0 0 0 1 0 0 0 0 1 0

图 5-3　位图索引结构图

在该索引图中，共有五类 JOB，每类 JOB 对应 14 个比特位（对应 14 行记录），其中某行在该列的值与 JOB 值对应则使用比特 1 表示，如 JOB = 'CLERK'，第一行在该列对应的值是 CLERK，就用比特 1 表示。否则用比特 0 表示，其他 JOB 类类似。

图 5-4 是位图索引操作的逻辑视图。通过该视图，读者可以更清楚了解 SQL 语句执行时，是如何使用位图索引的。

通过位图索引扫描 JOB='SALESMAN'对应的位图记录，找到值为 1 的行记录，即找到需要查找的数据。

下面给出一个例子来创建位图索引。

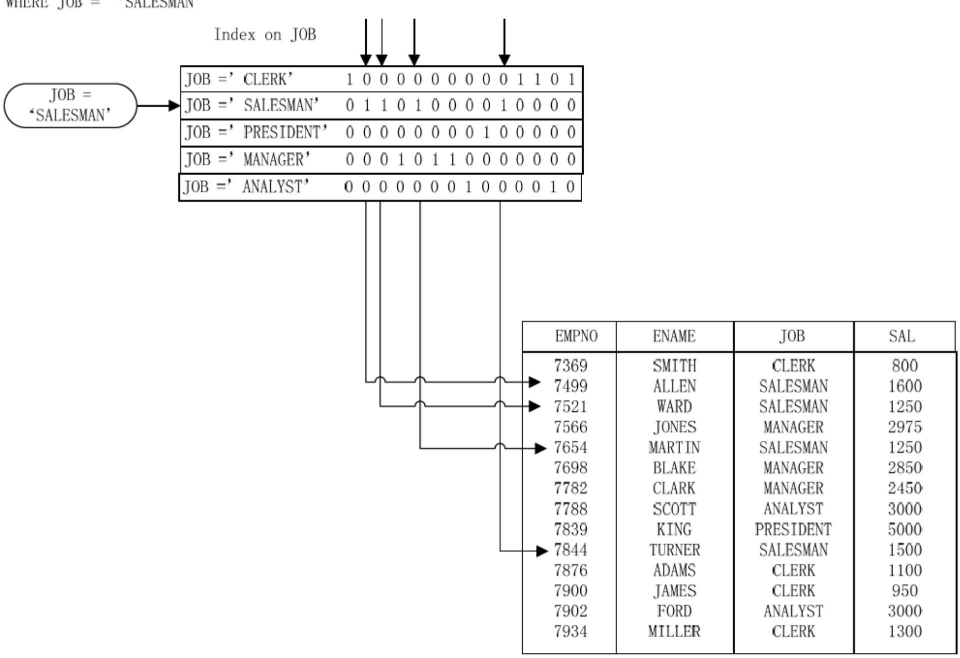


图 5-4　位图索引逻辑结构图

例子 5-61　创建位图索引。

```
SQL> create bitmap index emp_job_bitmap_idx
  2  on emp(job);

索引已创建。
```

此时，我们成功创建了位图索引 EMP_ENAME_BITMAP_IDX。该索引基于表 EMP 的 ENAME 列创建。我们通过下面例子查看该索引信息，主要关注其类型标识。

例子 5-62　查看位图索引。

```
SQL> col index name for a25
SQL> col index type for a15
SQL> col table name for a10
SQL>select index name,index type,table name,status
  2  from user indexes
  3* where index name like 'EMP%'

INDEX NAME              INDEX TYPE     TABLE NAME STATUS
--------------- ------------------- ------------- -----------------
EMP_JOB_BITMAP_IDX        BITMAP          EMP           VALID
```

输出说明创建的位图索引信息，其中 INDEX_TYPE 为 BITMAP，说明这是一个位图索引。

3．反向键索引

反向键索引是指在创建索引过程中，对索引列创建的索引键值的头尾调换后再存储，使用反向键索引的好处是将值连续插入到索引中时反向键能避免争用。

反向键索引适用于一种特殊的情形，如果一个索引值是按照序列值递增的，这样当连续插入大量数据时，所有的记录都将插入 B 树索引结构中的最右侧的叶子节点，并且会写入同一叶子节点中，这样难以避免产生争用问题而影响索引性能。正是为了避免这个问题引入了反向键索引。使用反向键索引使得每个键值按颠倒顺序，将序列性的键值分散开，使得键值平衡地保存在叶子节点中，如图 5-5 所示为键值颠倒的示意图。

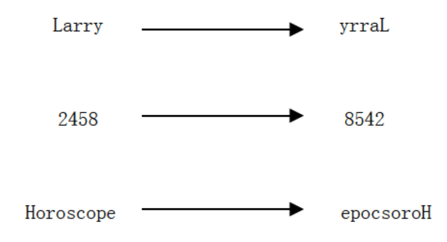


图 5-5　反向键索引的键值

创建反向键索引需要使用 REVERSE 关键字，如下例所示。

例子 5-63 创建反向键索引。

```
SQL> create index emp_sal_reverse_idx
  2  on emp(sal) reverse;

索引已创建。
```

同样，我们通过数据字典 USER_INDEXES 查看刚刚创建的反向键索引信息，读者需要注意 INDEX_TYPE 列的值，看 Oracle 是如何标识反向键索引类型的，如下例所示。

例子 5-64 通过数据字典视图查看反向键索引信息。

```
SQL> select index_name,index_type,table_name
  2   from user_indexes
  3  where index_name like 'EMP%';

INDEX_NAME                INDEX_TYPE            TABLE_NAME
-------------------- -----------------     ---------------------
EMP_ENAME_BITMAP_IDX     BITMAP                EMP
EMP_SAL_REVERSE_IDX      NORMAL/REV            EMP
```

4．基于函数的索引

在用户查询数据时，如果查询语句的 WHERE 字句中有函数存在，Oracle 使用函数索引将加快查询速度。基于函数的索引，使用表的列的函数值作为键值建立索引结构。

下面说明如何使用 UPPER 函数创建基于函数的索引。

例子 5-65　创建基于 UPPER 函数的函数索引。

```
SQL> create index dept dname idx
  2  on dept(UPPER(dname));

索引已创建。
```

如上例所示，我们创建了一个基于表 DEPT 中列 DNAME 的函数索引。创建该索引时，首先将列 DNAME 中的值转换成大写，然后对大写的 DNAME 创建索引，放入索引表中。这样当用户需要如下的查询时：

```
SELECT UPPER(DNAME) FROM DEPT WHERE UPPER(DNAME) ='NEW YORK'
```

Oracle 就不必对 WHERE 子句的条件做转化并逐行检索，对于选择的结果也不必使用 UPPER 函数再做转换的计算。显然此时使用基于函数的索引会极大的提高查询速度，如果该表很大的话，性能的提高是很明显的，如下例所示。

例子 5-66 通过数据字典 USER_INDEXES 查看基于函数的索引信息。

```
SQL> col index_type for a30
SQL> run
  1  select index_name,index_type,table_name
  2  from user_indexes
  3* where index_name like 'DEPT%'

INDEX NAME           INDEX TYPE                    TABLE NAME
-------------------- ---------------------------- --------------------
DEPT_DNAME_IDX        FUNCTION-BASED NORMAL          DEPT
```

上述输出中，同样读者需要注意 INDEX_TYPE 的值来区分索引类型，索引 DEPT_DNAME_IDX 的类型 INDEX_TYPE 为 FUNCTION_BASED NORMAL，说明它是基于函数的正常索引，该索引是在表 DEPT 上创建的。我们再使用下面例子查看该索引对应列的信息。

例子 5-67 通过数据字典 USER_IDX_COLUMNS 查看基于函数的索引信息。

```
SQL>  select index_name,table_name,column_name
  2   from user_ind_columns;

INDEX_NAME                TABLE_NAME  COLUMN_NAME
------------------------ ----------- --------------
PK_DEPT                   DEPT        DEPTNO
PK EMP                    EMP         EMPNO
EMP_ENAME_BITMAP_IDX      EMP         ENAME
EMP SAL REVERSE IDX       EMP         SAL
DEPT_DNAME_IDX            DEPT        SYS_NC00004$
```

从上面例子的输出可以看出，索引 DEPT_DNAME_IDX 的 COLUMN_NAME 为系统赋予的一个值，因为这个列不是 Oracle 可以使用明确的列名标识的，它是某个列的函数，所以 Oracle 就使用自己的方法来标识 GOLUMN_NAME 列明。

5．分区索引

如果一个基表是分布在多个分区上的，即该表是分区表，则对表建立的索引和表分区一一对

应，即每个分区表的索引和该分区表在同一个分区上存储，这样在数据分区和索引分区之间存在对应关系。这就是本地分区索引的特点。

使用本地分区索引使得索引和该索引所对应的数据分区分布在同一个分区中，使得索引和基表得以均匀分布。如果修改了基表分区，则 Oracle 自动维护对应的索引分区。创建本地分区索引时，我们需要对一个分区表建立本地分区索引，对于普通表则没有意义。这里假设表 products 为分区表，对该表的列 PRO_DATE 生产日期建立索引，如下例所示。

例子 5-68 创建本地分区索引。

```
SQL> create index products pro date idx
  2  on products(pro date) local
  3  tablespace product_tbs;
```

和本地分区索引对应的是全局分区索引，全局分区索引可以分区也可以不分区，索引和基表分区之间不存在一一对应关系。全局分区索引与本地表的分区不一致。如果使用索引的目的是对表的查询，则使用本地分区索引。

下面再介绍两种分区表的索引类型，即前缀索引和非前缀索引，前缀索引对索引列上的一个左前缀进行分区的索引，而非前缀索引不包含对索引列的左前缀进行分区的索引。如果查询的列不是分区表的键的一部分，则最好使用全局前缀索引。如果使用并行查询，则最好选择使用本地非前缀索引。

6．索引组织表

索引组织表简称为 IOT（Index-Organized Table）。首先它是一个表，而不是索引，把它放在索引类型中介绍主要是因为索引组织表具有索引的特性。实际上 IOT 使用 B 树索引结构来存储数据的表，它包含排序的数据以及键值，即 IOT 使用 B 树索引结构存储对主键排序过的数据。

在对 IOT 表进行查询时，实际的行数据以及被索引的列值都保存在索引叶块中，所以一次 I/O 即读取所需的数据，而不需要向普通的堆组织表一样先读取一次索引，再通过 ROWID 读取数据。IOT 表适合于大的数据库或者 OLTP 应用系统，下面我们创建一个索引组织表。

首先创建一个表空间，其作用我们在创建完索引组织表之后再讨论，如下例所示。

例子 5-69 创建存储溢出的索引组织表数据的表空间。

```
SQL> create tablespace ovrflwtbs
  2  datafile 'e:/oracle/product/10.2.0/oradata/lszpri/ovrflw.dbf'
  3  size 100m;

表空间已创建。
```

然后，创建索引组织表，如下例所示。

例子 5-70 创建索引组织表。

```
SQL> create table student_info(
  2  student_id  number,
  3  student_name varchar2(20),
  4  student_major varchar2(30),
  5   student_add  varchar2(50),
  5   constraint pk_student_id primary key (student_id))
```

```
  7  organization index tablespace users        #说明是索引组织表，表空间为 USERS
  8  pctthreshold 20                            #索引块中位为 IOT 表预留的空间百分比
  9  overflow tablespace ovrflwtbs;             #索引块中溢出的数据保存在该表空间

表已创建。
```

在创建索引组织表时，我们使用了关键字 PCTTHRESHOLD，其值为 20，意思是索引块只有 20%的空间用于存储索引项。在索引组织表中因为索引项可能很大，所以设计该索引块只使用自己一定比例空间，比如 20%的索引块空间，将该行的其他部分保存在另一个表空间中，此时就使用 OVERFLOW 指定的表空间来存储该行溢出的数据。



如果此时的 PCTTHRESHOLD 设置的太小会造成较多数据行都有溢出数据保存在其他表空间中。这样，显然查询该行数据就需要多表空间之间的数据连接，影响查询性能。

5.8.5 使用绑定变量

在介绍使用绑定变量来优化 SQL 语句之前，我们了解一下解析 SQL 语句相关的 Oracle 数据库组件。在 Oracle 数据库体系结构中的 SGA 包含共享池组件，共享池就是存放解析后的 SQL 语句，此时的共享池包含 SQL 语句的最终执行计划，如果有相同的 SQL 查询语句，就不需要再次解析 SQL 语句，而是直接从共享池中执行 SQL 语句的执行计划，这也是共享池的作用，即共享 SQL 代码。

我们已经知道 SQL 语句的解析分为软解析和硬解析，使用共享池就是避免硬解析的发生，因为硬解析需要重新分析 SQL 语句的语法语义，然后通过 CBO 优化生成最终的执行计划，这样就很消耗 CPU 时间。

对于两个 SQL 语句，Oracle 通过正文来判断二者是否相等。对于相等的 SQL 语句，Oracle 可以重用共享池中对应 SQL 语句的执行计划，下面给出两个不相等的 SQL 语句例子。

SQL 语句一：

```
SQL> select empno,ename,job,sal
  2  from scott.emp
  3  where deptno=20;
```

SQL 语句二：

```
SQL> select empno,ename,job,sal
  2  from scott.emp
  3  where deptno=30;
```

虽然上述两个 SQL 语句只有谓词 where deptno 的值不同，但是 Oracle 认为这两个 SQL 语句是不相等的。所以，如果系统的编码过程中有大量类似的查询语句出现，一定要使用绑定变量，这样可以极大减少 SQL 语句的硬解析时间，如下例所示。

例子 5-71 使用绑定变量的 SQL 查询语句。

```
SQL> select empno,ename,job,sal
  2  from scott.emp
  3  where deptno=&deptno;
```

此时，使用绑定变量&deptno 就使得 SQL 代码可以共享。当然共享的 SQL 代码的数量又涉及到共享池的大小，如果内存不是问题，可以尽量将共享池设置的大些，但是共享池的尺寸又受到 SGA 的限制。这是个相互关联的问题，需要读者在调整时注意。

5.8.6 消除子查询优化 SQL 语句

我们可以使用 AUTOTRACE 分析存在子查询的 SQL 语句的执行结果，并对 SQL 语句进行优化。

我们给出一个例子，对查询用户 SCOTT 的 EMP 表进行嵌套子查询，分析该语句的执行过程。先来分析一个子查询的例子。

例子 5-72 对查询用户 SCOTT 的 EMP 表进行嵌套子查询。

```
SQL> select *
  2  from scott.emp e1
  3  where e1.sal>
  4  (select avg(sal)
  5   from scott.emp e2
  5   where e2.deptno = e1.deptno);
```

在上述 SQL 查询语句中，每扫描表 E1 中的每一行，然后执行一次子查询，这样如果表 E1 有 N 行，而自查也要执行 M 行，则每次执行需要执行 N*M 次操作。下面我们启动 AUTOTRACE 跟踪该 SQL 执行语句。

例子 5-73 开启 AUTOTRACE 功能。

```
SQL> set autotrace traceonly
```

下面跟踪 sql 语句的执行。

例子 5-74 跟踪 SQL 语句的执行。

```
SQL> select *
  2  from scott.emp e1
  3  where e1.sal>
  4  (select avg(sal)
  5   from scott.emp e2
  5   where e2.deptno = e1.deptno);


执行计划
----------------------------------------------------------
Plan hash value: 2849554444

-------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     2 |   174 |     5   (0)| 00:00:01 |
|*  1 |  FILTER              |      |       |       |            |          |
|   2 |   TABLE ACCESS FULL | EMP  |    12 |  1044 |     3   (0)| 00:00:01 |
|   3 |   SORT AGGREGATE    |      |     1 |    28 |            |          |
```

```
|* 4 |    TABLE ACCESS FULL| EMP  |    1 |   28 |    3   (0)| 00:00:01 |
--------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("E1"."SAL"> (SELECT AVG("SAL") FROM "SCOTT"."EMP" "E2"
              WHERE "E2"."DEPTNO"=:B1))
   4 - filter("E2"."DEPTNO"=:B1)

Note
-----
   - dynamic sampling used for this statement


统计信息
----------------------------------------------------------
         53  recursive calls
          0  db block gets
         95  consistent gets
          0  physical reads
          0  redo size
       1007  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

上述 SQL 语句的执行迭代次数多，数据的一致性读次数多，随着表的增大会呈现出指数增长，极大地消耗 CPU 资源。

下面改写 SQL 语句，使用联机视图改写子查询，继续使用 AUTOTRACE 分析。

例子 5-75 跟踪改写的 SQL 语句。

```
SQL> select *
  2  from emp e1,(select e2.deptno deptno,avg(e2.sal) avg sal
  3            from scott.emp e2
  4            group by deptno) dept avg sal
  5  where e1.deptno = dept avg sal.deptno
  5* and e1.sal > dept avg sal.avg sal


执行计划
----------------------------------------------------------
Plan hash value: 2230095557

--------------------------------------------------------------------------
| Id  | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    2 |   228 |     8  (28)| 00:00:01 |
|*  1 |  HASH JOIN         |      |    2 |   228 |     8  (28)| 00:00:01 |
|   2 |   TABLE ACCESS FULL | EMP |   12 |  1044 |     3   (0)| 00:00:01 |
```

```
|   3 |   VIEW              |      |    12 |   312 |     4  (28)| 00:00:01 |
|   4 |    HASH GROUP BY    |      |    12 |   312 |     4  (28)| 00:00:01 |
|   5 |     TABLE ACCESS FULL| EMP  |    12 |   312 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("E1"."DEPTNO"="DEPT AVG SAL"."DEPTNO")
       filter("E1"."SAL">"DEPT_AVG_SAL"."AVG_SAL")

Note
-----
   - dynamic sampling used for this statement


统计信息
---------------------------------------------------
         20  recursive calls
          0  db block gets
         52  consistent gets
          0  physical reads
          0  redo size
       1212  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          4  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

使用联机视图重新编写的 SQL，使得该 SQL 语句的查询过程只需要 N+M 次操作即可完成。显然其伸缩性增强，因为计算不会呈指数增长，从输出看出迭代的 CALLS 在减少，而且数据的一致性 GETS 也减少，随着表 EMP 的数据量的增加，修改后的查询会极大的减少 CPU 的消耗。

5.9 被动优化SQL语句

在程序打包后，或者系统运行后如何优化 SQL 语句？当然优化的前提是需要优化，比如系统运行缓慢、发现了导致等待事件的 SQL 语句等，此时就需要分析该 SQL 语句的语法结构，给出合理的优化建议，一般是建立或删除索引、建立分区表等操作。下面介绍如何完成被动优化 SQL 语句。

5.9.1 使用分区表

分区表是将一个大的表按照一定的规则，分解成几个逻辑结构相同的表。在查询时，用户使用相同的表名字，但是 Oracle 会根据具体的查询条件选择分区表的特定分区部分，因为分区规则限制了查询的数据量，这样就减少了查询大表的时间。

如果在生产数据库中由于一个表在实际使用中快速上升到上千万行的数据，此时的查询性能

就受到极大的影响，即使使用索引，性能也很难满足用户需求。此时就需要考虑根据时间或者范围以及组合条件来创建分区表。

5.9.2　使用表和索引压缩

为了增强 SQL 语句的查询性能，可以对表、索引以及物化视图进行压缩，压缩后的表会根据表中重复数据的多少减少存储空间，以减少查询时间。

实际上表压缩思想很简单，即将表中重复的数据去掉，采用算法来替换这些重复的值，在需要时通过算法重建这些重复的数据，从而实现对表的压缩。下面我们通过例子来看如何创建压缩表。

例子 5-76　创建压缩表。

```
SQL> create table compress_emp
  2  compress
  3  tablespace users
  4  as
  5  select *
  5  from scott.emp;

表已创建。
```

下面，我们通过数据字典 USER_TABLES 查询表 COMPRESS_EMP 是否启动压缩。

例子 5-77　查询是否成功创建压缩表 COMPRESS_EMP。

```
SQL> select table name,tablespace name,compression
  2  from user tables
  3  where table name like 'COMPRESS%';

TABLE NAME                TABLESPACE NAME          COMPRESS
------------------------- ------------------------------ ----------------
COMPRESS_EMP               USERS                      ENABLED
```

同样，我们可以创建一个压缩索引，如下例所示。

例子 5-78　创建压缩索引。

```
SQL> create index compress_emp_ename_idx
  2  on compress_emp(ename)
  3  compress;

索引已创建。
```

压缩创建的索引去掉了索引中的重复值，尤其对于大表，可以减少存储空间并增强查询性能。

5.9.3　保持 CBO 的稳定性

在版本升级或者应用程序升级后的 CBO 的执行策略会发生变化，使得 CBO 因为环境的变化而改变计划的执行，所以此时就需要某种方式来确保 CBO 执行计划的稳定性，这种方式就是“存储大纲”，它维持 CBO 的计划执行稳定性。只要是相同的查询 SQL 语句，都使用相同的执行计划。

1．创建存储大纲的前提

要创建存储大纲必须保证两个初始化参数在所有环境下的一致性，同时将这两个参数 QUERY_REWRITE_ENABLED 以及 STAR_TRANSFORMATION_ENABLED 设置为 TRUE，并且保证参数 OPTIMIZER_FEATURES_ENABLE 在所有环境下一致。下面我们查询这三个参数的值，如下例所示。

例子 5-79 验证系统是否具备创建存储大纲的前提。

```
SQL> show parameter query rewrite enabled;

NAME                               TYPE        VALUE
-------------------------------- ---------- --------------------------
query_rewrite_enabled                   string      TRUE
SQL> show parameter star transformation enabled;

NAME                               TYPE        VALUE
-------------------------------- ---------- --------------------------
star_transformation_enabled             string      TRUE
SQL> show parameter optimizer_features_enable;

NAME                               TYPE        VALUE
-------------------------------- -—-------- --------------------------
optimizer_features_enable               string      10.2.0.1
```

通过查询，我们知道当前的系统具备创建存储大纲的条件。下面我们介绍如何创建存储大纲。

2．创建存储大纲

Oracle 允许在不同级别上创建存储大纲，如在数据库级别、会话级别以及当前 SQL 语句级别。创建的存储大纲信息保存在 OL$表中，该表默认保存在 SYSTEM 表空间中。我们下面分别演示如何创建不同级别的存储大纲以及注意事项。

（1）创建数据库级的存储大纲

创建数据库级存储大纲的含义是对当前数据库上执行的每一个 SQL 语句都创建一个存储大纲，对于非常大的数据或者 OLTP 系统，这样会创建很多的存储大纲，对于 SYSTEM 表空间是一个压力，所以最好为表 OL$创建特定的表空间。要创建数据库级存储大纲需要将参数 CREATE_STORED_OUTLINES 设置为 TRUE，如下例所示。

例子 5-80 创建数据库级存储大纲。

```
SQL> alter system set create_stored_outlines = true;

系统已更改。
```

（2）创建会话级的存储大纲

这里，我们使用 ALTER SESSION 命令对当前的会话创建存储大纲，这样对于当前会话后执行的所有 SQL 语句都将创建一个存储大纲，如下例所示。

例子 5-81　在会话级创建存储大纲。

```
SQL> alter session set create stored outlines=true;

会话已更改。
```

（3）为特定 SQL 语句创建存储大纲。

使用 CREATE OUTLINE 语句为特定的 SQL 语句创建存储大纲，如下例所示。

例子 5-82　为 SQL 语句创建存储大纲。

```
SQL> create outline emp_outline
  2  on
  3  select *
  4  from scott.emp
  5  tablespace oltbs;

大纲已创建。
```

如上，我们为 SQL 语句 select * form scott.emp 创建了存储大纲，大纲名为 emp_outline。下面我们通过数据字典 OL$查询为大纲 EMP_OUTLINE 的创建信息。此时需要使用 OUTLN 用户登录数据库，因为该用户才拥有数据库中的存储大纲，大纲信息记录在表 OL$中。如下例所示，使用 OUTLN 用户登录数据库。

例子 5-83　使用用户 OUTLN 登录数据库。

```
SQL> conn outln/oracle
已连接。
```

然后，通过表 OL$查询存储大纲 EMP_OUTLINE 的创建信息，如下例所示。

例子 5-84　查询 EMP_OUTLINE 创建信息。

```
SQL>select ol_name,sql_text,creator,timestamp
  2  from ol$
  3* where ol_name like 'EMP%'

OL_NAME     SQL_TEXT                 CREATOR     TIMESTAMP
----------- ------------------------ -------------------- ---------
EMP_OUTLINE     select *                 SYS        31-5 月 -10
              from scott.emp
              tablespace oltbs
```

可见在表 OL$中，清晰地记录了大纲 EMP_OUTLINE 对应的 SQL 语句。



如果在数据库级创建存储大纲，则大纲名称是由 Oracle 自己创建，不需要用户干预。

下面我们查询这些大纲的名字，如下例所示。

例子 5-85　查询 Oracle 自动产生的存储大纲的名字。

```
SQL> set line 120
SQL> run
```

```
  1  select ol_name,sql_text
  2* from ol$

OL_NAME                      SQL_TEXT
-------------------------  ---------------------------------------------
SYS OUTLINE 10052911244500001  select sysdate + 1 / (24 * 50) from dual
SYS_OUTLINE_10052911255090513  select sysdate + 1 / (24 * 50) from dual
EMP_OUTLINE                    select *
                          from scott.emp
                          tablespace oltbs
SYS OUTLINE 10052911312415525  SELECT * FROM RLM$JOBQUEUE WHERE
SCHED AT < SYSTIM
                          ESTAMP AND ROWNUM < 2 FOR UPDA
SYS_OUTLINE_10052911312417125  SELECT MIN(SCHED_AT) FROM RLM$JOBQUEUE
```

以上输出中粗体字的内容都是 Oracle 自动创建的存储大纲名称。如果不需要该存储大纲，可以删除。

3．删除存储大纲

可以使用 DROP OUTLINE 指令删除，如下例所示。

例子 5-86 删除存储大纲。

```
SQL> drop outline emp outline;
drop outline emp outline
*
第 1 行出现错误：
ORA-18005：此操作需要 DROP ANY OUTLINE 权限
```

上述删除存储大纲的操作失败，从错误提示清楚地知道当前的用户不具备权限，因为当前的用户是 OUTLN，所以我们切换到 SYS 用户继续删除大纲操作，如下例所示。

例子 5-87 在 SYS 用户下删除存储。

```
SQL> conn sys/oracle as sysdba
已连接。
SQL> drop outline emp_outline;

大纲已删除。
```

4．启用存储大纲

Oracle 不会自动使用存储大纲，需要手工设置参数 USE_STORED_OUTLNES 为 TRUE，Oracle 才会使用 ALTER SYSTEM ALTER SESSION 创建的存储大纲。如下例所示，修改初始化参数 USE_STORED_OUTLINES 为 TRUE。

例子 5-88 修改参数 USE_STORED_OUTLINES 为 TRUE。

```
SQL> alter system set use_stored_outlines=true;

系统已更改。
```

5.9.4　创建合适的索引

合理使用索引既是主动的 SQL 优化，也是被动的 SQL 优化。如果在设计初期考虑优化 SQL 语句，从而创建合适的索引就属于主动的 SQL 优化；如果在系统部署之后，运行中发现某个 SQL 语句的执行影响系统性能，出于性能的考虑，此时通过创建合适的索引来优化 SQL 语句的执行就是被动的 SQL 优化。具体方法，我们在 5.4.4 节使用索引中已经介绍过，这里就不再介绍。

5.10　详解V$SQL视图

在 SQL 的优化中，V$SQL 视图提供了详细的 SQL 代码执行结果信息，通过该视图可以判断效率低下或者耗费资源过度的 SQL 语句。该视图保留实例启动以来的所有 SQL 语句，但是出于空间原因有可能会删除旧的 SQL 语句。

下面我们分析通过 V$SQL 视图如何定位耗费不同资源的 SQL 语句。

1．查询消耗磁盘 I/O 最多的 SQL 语句

我们知道内存读取的速度比磁盘读取数据的速度要快的多。为了提高数据的读取效率，希望用户查询的数据在 SGA 中，但是共享池的大小有限，此时就不可回避对磁盘数据的读取。在 V$SQL 视图中，列 disk_reads 说明磁盘读取的数量，我们对其进行排序以判定磁盘 I/O 最多的 SQL 语句。

例子 5-89　查询自实例启动以来磁盘 I/O 最多的 SQL 语句。

```
SQL>select sql text,executions,disk reads
  2   from v$sql
  3  where disk reads>&number
  4* order by disk reads desc
输入 number 的值:  400
原值    3: where disk reads>&number
新值    3: where disk reads>400

SQL TEXT                                             EXECUTIONS DISK READS
--------------------------- ------------------- -------------------------
select /*+ index(idl ub1$ i idl ub11) +*/ pie            435        794
ce#,length,piece from idl ub1$ where obj#=:1
and part=:2 and version=:3 order by piece#

DECLARE job BINARY INTEGER := :job; next date            1224        418
DATE := :mydate;  broken BOOLEAN := FALSE; B
EGIN EMD MAINTENANCE.EXECUTE EM DBMS JOB PROC
S(); :mydate := next_date; IF broken THEN :b
:= 1; ELSE :b := 0; END IF; END;
```

在以上输出中，我们使用变量&number 使得用户可以自定义磁盘读取的次数。在输出中，我们使用了 EXECUTIONS 列，该列说明语句执行的时间量，包括该语句的等待时间和服务时间。

2．分析缓冲区读取次数最多的 SQL 语句

缓冲区获取即 buffer_gets，高的缓冲区获取说明该 SQL 语句耗费较多的 CPU 资源，降低逻辑

读取是 SQL 优化的重要方面，逻辑读取分为 DB 块读取和一致性读取。前者是从数据库高速缓冲区读，后者是通过 UNDO 回滚段读取，大量的缓冲区读取意味着 CPU 在高速计算。

例子 5-90 查询逻辑读取数最多的 SQL 语句。

```
SQL>select sql text,buffer gets,parse calls
  2   from v$sql
  3   where buffer gets>&number
  4* order by buffer gets
输入 number 的值:  20000
原值    3:  where buffer gets>&number
新值    3:  where buffer_gets>20000

SQL TEXT                                                  BUFFER GETS PARSE CALLS
---------------------------------- ----------- --------------------------
select /*+ rule */ bucket_cnt, row_cnt, cache        22394        228
_cnt, null_cnt, timestamp#, sample_size, mini
mum, maximum, distcnt, lowval, hival, density
, col#, spare1, spare2, avgcln from hist_head
$ where obj#=:1 and intcol#=:2

select job, nvl2(last_date, 1, 0) from sys.jo        22837          1
b$ where (((:1 <= next_date) and (next_date <
 :2))    or  ((last_date is null) and (next_d
ate < :3))) and (field1 = :4 or (field1 = 0 a
nd 'Y' = :5)) and (this_date is null) order b
```

通过上述查询，可以确定逻辑读取最多的 SQL 语句，但是并不是高逻辑读取数就意味着该 SQL 语句影响性能，只有系统确认出现性能问题，且通过该视图给出可能的影响数据库性能的 SQL 语句。需要仔细分析这些语句的其他信息，如通过 EXPLAI、AUTOTRACE 和 TKPROF 等工具对该语句的执行结果做更细致的分析。

例子 5-91 查询当前系统耗费 CPU 资源最多的前 5 个 SQL 语句。

```
SQL> select sql text,cpu time,buffer gets
  2  from (select sql text,cpu time,buffer gets
  3  from v$sql
  4  order by cpu time desc)
  5* where rownum<5

SQL TEXT                                              CPU TIME  BUFFER GETS
---------------------------------------------------- ---------- -----------
BEGIN EMD NOTIFICATION.QUEUE READY(:1, :2, :3); EN    7875743      82241
D;

begin MGMT_JOB_ENGINE.get_scheduled_steps(:1, :2,       7445531     137072
:3, :4); end;

insert into wrh$_sga_target_advice  (snap_id, dbi      5975582     175
d, instance_number,   SGA_SIZE, SGA_SIZE_FACTOR,
ESTD_DB_TIME, ESTD_PHYSICAL_READS) select   :sna
p_id, :dbid, :instance_number,   SGA_SIZE, SGA_SI
```

```
ZE_FACTOR, ESTD_DB_TIME, ESTD_PHYSICAL_READS  from

SQL_TEXT                                              CPU_TIME BUFFER_GETS
--------------------------------------------------- ---------- -----------
   v$sga_target_advice

DECLARE job BINARY_INTEGER := :job; next_date DATE   3894950     248085
 := :mydate;  broken BOOLEAN := FALSE; BEGIN EMD_M
AINTENANCE.EXECUTE_EM_DBMS_JOB_PROCS(); :mydate
:=next_date; IF broken THEN :b := 1; ELSE :b := 0;
END IF; END;

CALL MGMT_ADMIN_DATA.EVALUATE_MGMT_METRICS 1511189    53579
(:tguid, :mguid, :result)
```

我们查询了当前消耗 CPU 资源最多的几个 SQL 语句，按照降序排列，由于笔者当前的数据是测试数据库，所以查询的结果都是系统内部的 SQL 调用。

5.11　本章小结

SQL 优化是 Oracle 数据库优化的重要内容，Oracle 默认使用基于成本的优化方式，即 CBO 优化。CBO 优化的本质是通过分析数据库对象，如表、索引的数据、数据字典数据以及系统的 CPU 和 I/O 数据来选择一个 SQL 语句的最佳执行计划。这些数据的统计又分为自动数据统计和手工数据统计。

对于 SQL 的优化，只要条件允许，比如参与数据库系统的设计以及参与或指导编码，在早期阶段就优化 SQL 代码，比如对 WHERE 谓词都要建立索引、选择合理的联结方法、使用绑定变量并遵循主动 SQL 调整的其他原则。被动的 SQL 调整也是 DBA 常见的一种优化环境，因为很多 DBA 在参与工作时，系统已经部署完毕，此时就需要通过等待事件分析、V$SQL 视图的使用，确定消耗资源最多的 SQL 语句，这里的资源取决于用户的关注角度，比如是 CPU、I/O，还是内存使用等。

第 6 章

◂ Oracle实例优化 ▸

Oracle 实例是由内存组件和相关的后台进程组成，这些内存组件提高了数据库系统的运行，而后台进程负责管理系统或者内存组件，使得整个数据库系统协调一致地工作。但是 Oracle 数据库的实例并不满足所有的生产数据库系统的需求，即使使用自动内存管理，Oracle 也只是提出一个建议。本章我们讨论在 Oracle 实例优化中最典型的 SGA 组件优化方法。

6.1 详解SGA与实例优化

Oracle 的 SGA 是指系统全局区，它是数据库运行期间使用的一段公有内存，即所有使用数据库的用户都可以访问这部分内存，它包括共享池、重做日志缓冲区、数据库缓存高速缓冲区、java 池、大池以及流池组成。

这些组件是实例优化的操作对象，因为优化 SGA 就是调整这些数据库组件的参数，从而提高系统的运行效率，如提高用户查询的响应事件等。SGA 的组成如图 6-1 所示。

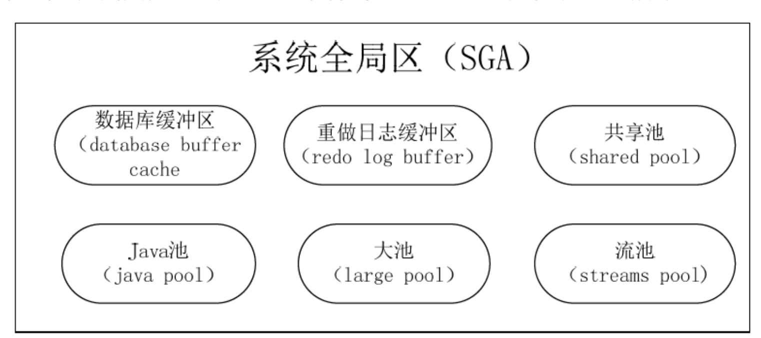


图 6-1 Oracle 的 SGA 组成图

这里我们解释图 6-1 中各个组件的作用以及涉及的参数，这样读者在修改上述组件时就更有针对性，做到“有的放矢”。

- 数据库缓冲区：该区域存放用户从数据库中读取的数据，在用户查找数据库时会首先在数据库缓存中搜索，如果没有才会读数据库文件，所以该区域不能设置的过小，不然频繁的读取数据文件会增加查询时间，因为磁盘 I/O 是耗时的行为。
- 重做日志缓冲区：该缓冲区放置用户改变的数据，所有变化了的数据和回滚需要的数据都

暂时保存在重做日志缓冲区中。涉及的参数为 log_buffer，如下例所示我们查询重做日志缓冲区的大小。

例子 6-1 查询重做日志缓冲区的大小。

```
SQL> show parameter log buffer;

NAME                                 TYPE           VALUE
------------------------------------ -------------------- ----------------
log_buffer                           integer        6024640
```

- 共享池：共享池包括数据字典高速缓存和库高速缓存，库高速缓存存放 Oralce 解析的 SQL 语句、PL/SQL 过程、包以及各种控制结构如锁、库缓冲句柄等。而数据字典高速缓存保存执行 sql 语句所需的各种数据字典定义，如表和列的定义、用户访问表的权限等。
- Java 池：执行 java 代码的区域。它为 Oracle 数据库中运行的 JVM（JAVA 虚拟机）分配的一段固定大小的内存。
- 大池：该内存区提供大型的内存分配，在共享服务器连接模式下提供会话区。在使用 RMAN 备份时，也使用该内存区作为磁盘 I/O 的数据缓冲区。
- 流池：该区域称为流内存，是为 Oracle 流专用的内存池，流是 Oracle 数据库中的一个数据共享，其大小可以通过参数 STREAMS_POOL_SIZE 动态调整。

在 Oracle 11g 以及更高版本中，SGA 的中内存参数可以动态修改，但是总的内存大小受到参数 SGA_MAX_SIZE 的限制。在安装数据库时，这个参数的值是默认的，而实际的生产数据库往往需要重新设置一个新值，以利用操作系统中充足的内存资源。查看参数 SGA_MAX_SIZE 的值，如下例所示。

例子 6-2 查看参数 SGA_MAX_SIZE 的值。

```
SQL> show parameter sga max size;

NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ------------
sga_max_size                         big integer          576M
```

我们继续查看 SGA 信息，如下例所示，查看当前数据库的 SGA 信息。

例子 6-3 查看 SGA 信息。

```
SQL> show sga;

Total System Global Area   603979776 bytes
Fixed Size                  170380 bytes
Variable Size             222301108 bytes
Database Buffers          373293056 bytes
Redo Buffers                7135232 bytes
```

上述输出的第一个参数 Total System Global Area 其实和例子 6-2 中查到的是同一个数据，二者大小相等，如下例所示。

例子 6-4 计算 SGA 值。

```
SQL> select 576*1024*1024 bytes
  2  from dual;

     BYTES
-----------------
 603979776
```

下面我们调整 SGA 的最大尺寸，目的是增大 Oracle 在整个内存中所占的比例，但是不能太大，一般可以设置为当前内存大小的一半即可。通过修改参数 SGA_MAX_SIZE 以修改 SGA 的尺寸，如下例所示。

例子 6-5 修改 SGA_MAX_SIZE 参数。

```
SQL> alter system set sga max size= 700M scope = spfile;

系统已更改。
```

上例中，我们把 SGA_MAX_SIZE 改为 700M。下面我们查询这次修改，如下例所示。

例子 6-6 查询参数 SGA_MAX_SIZE 修改结果。

```
SQL> show parameter sga_max_size;

NAME                                TYPE          VALUE
------------------------------ ------------- -------
sga_max_size                         big integer   576M
```

观察 VALUE 的值发现该值为 576M，没有变化，这里需要向读者说明参数 SGA_MAX_SIZE 是静态参数，需要重启数据库后方可生效。我们先不关闭数据库，继续对 SGA 优化。

接下来查看在 Oracle 的静态参数中有哪些和 SGA 相关，如下例所示。

例子 6-7 查看和 SGA 相关的静态参数。

```
SQL> show parameter sga;

NAME                                TYPE          VALUE
---------------------------- -------------- -------------
lock_sga                            boolean        FALSE
pre_page_sga                        boolean        FALSE
sga_max_size                        big integer    576M
sga_target                          big integer     576M
```

下面我们依次介绍参数 LOCK_SGA、PRE_PAGE_SGA 和 SGA_TARGET 看这些参数对优化 SGA 有什么作用。

1．LOCK_SGA 的含义及优化

该参数的作用是将 SGA 锁定（lock）在物理内存内，这样就不会出现 SGA 使用虚拟内存的情况，显然这样可以提高数据的读取速度，记住磁盘 I/O 操作永远要尽量避免或减少。该参数的默认值为 FALSE，即不将 SGA 锁定在内存中。下面，我们修改参数 LOCK_SGA 为 TRUE，如下例所示。

例子 6-8 设置参数 LOCK_SGA 为 TRUE。

```
SQL> alter system set lock sga = true scope = spfile;

系统已更改。
```

该参数是静态参数，需要重启数据库才可生效。

2．PRE_PAGE_SGA 的含义及优化

该参数的作用是启动数据库实例时，将整个 SGA 读入物理内存，对于内存充足的系统而言，这样显然可以提高系统运行效率。我们修改该参数为 TRUE，如下例所示。

例子 6-9 设置参数 PRE_PAGE_SGA 为 TRUE。

```
SQL> alter system set pre page sga= true scope = spfile;

系统已更改。
```

下面我们关闭数据库并重启数据库，如下例所示。

例子 6-10 关闭并重启数据库。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  734003200 bytes
Fixed Size                 171172 bytes
Variable Size            356518044 bytes
Database Buffers         369098752 bytes
Redo Buffers               7135232 bytes
数据库装载完毕。
数据库已经打开。
```

现在，我们查看刚才修改的三个参数：SGA_MAX_SIZE、LOCK_SGA、PRE_PAGE_SGA，如下例所示。

例子 6-11 查看与 SGA 相关的参数修改结果。

```
SQL> show parameter sga;

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------
lock sga                             boolean     TRUE
pre_page_sga                         boolean     TRUE
sga max size                         big integer 700M
sga_target                           big integer  576M
```

从上面输出可以看出参数 LOCK_SGA 和 PRE_PAGE_SGA 的值都为 TRUE，参数 SGA_MAX_SIZE 的值也修改为 700M。读者或许会问，参数 SGA_TARGET 起什么作用呢？

3．SGA_TARGET 的含义及优化

在 Oracle10g 以及以上的版本中，提供了内存的自动管理，这样 Oracle 可以根据业务需要和服务器自身的软硬件环境自动调整一些内存参数。参数 SGA_TARGET 就决定是否使用 SGA 自动管理，该参数的默认值和系统的 SGA_MAX_SIZE 一样大，当该参数值不为 0 时，则启动 SGA 的自动管理，该参数可以动态修改。下面我们修改该参数的值为 700M，如下例所示。

例子 6-12 修改参数 SGA_TARGET 的值。

```
SQL> alter system set sga_target = 700M;

系统已更改。
```

读者可以自行查看修改结果，这里不再给出查询结果。

既然 SGA 可以自动管理，但不是所有的内存组件都可以自动调整，那么哪些 SGA 的内存可以自动调整呢，下面是 SGA 可以自动调整的内存组件。

- 共享池。
- java 池。
- 大池。
- 数据库缓冲区。
- 流池。

这些组件的尺寸不需要用户干预，其值自动设置为 0。我们使用视图 v$parameter 查看这些自动调整的内存组件的信息。

例子 6-13 查看自动自动调整的内存组件的信息。

```
SQL> col name for a30
SQL> col value for a20
SQL>select name,value,isdefault
  2  from v$parameter
  3  where name in ('shared_pool_size','large_pool_size',
  4* 'java_pool_size')

NAME                          VALUE              ISDEFAULT
-------------------- ----------------- -----------------
shared_pool_size                0                 TRUE
large_pool_size                 0                 TRUE
java_pool_size                  0                 TRUE
```

虽然这些参数是可以自动调整的，但是用户依然可以使用 ALTER SYSTEM SET 指令修改内存组件的尺寸。如下例所示，修改 JAVA_POOL_SIZE 的值为 10M。

例子 6-14 修改 JAVA_POOL_SIZE 的值。

```
SQL> alter system set java pool size = 10M;
```

```
系统已更改。
```

下面我们查询修改结果，如下例所示。

例子 6-15　查询参数 JAVA_POOL_SIZE 的修改结果。

```
SQL> show parameter java pool size;

NAME                                 TYPE            VALUE
------------------------------------ --------------- ----------
java_pool_size                       big integer      12M
```

显然参数 JAVA_POOL_SIZE 的值不再是 0，其值变为 12M。读者或许会问在例子 6-14 中设置该参数值为 10M，怎么变成 12M 呢。其实是 Oracle 会正对系统自身情况做一些调整，是数据库自己的行为，读者不必太在意。

6.2　将程序常驻内存

在 Oracle 数据库中有一个软件包 DBMS_SHARED_POOL，它提供过程 KEEP 和 UNKEEP，将用户经常使用的程序如存储过程、触发器、序列号、游标以及 JAVA SOURCE 等数据库对象长期保存在一个内存结构中，这个内存区就是共享池（shared pool）。对于用户频繁使用的这些数据库对象而言，将其常驻内存可以减少磁盘 I/O 从而减少用户的响应时间。本节我们先讲解几个数据块缓冲池，分别解释他们的作用以及使用时机，然后介绍如何将一个存储过程常驻内存。最后介绍创建软件包 DBMS_SHARED_POOL 的 dbmspool.sql 过程，从而更清楚地理解软件包中各种过程的作用以及参数含义。

6.2.1　创建软件包 DBMS_SHARED_POOL

在 Oracle 数据库中软件包 DBMS_SHARED_POOL 不是默认安装的，所以需要执行一个 sql 脚本文件来创建该软件包，它有两个经常使用的过程 KEEP 和 UNKEEP，KEEP 过程实现将程序常驻内存，而 UNKEEP 过程将指定的程序清除出内存。如果读者开始就尝试执行软件包 DBMS_SHARED_POOL 的 KEEP 过程，则会提示如下例所示的错误。

首先使用 DBA 用户登录数据库。

例子 6-16　登录数据库并执行 KEEP 过程。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> execute dbms_shared_pool.keep('HR.SECURE_DML');
BEGIN dbms_shared_pool.keep('HR.SECURE_DML'); END;

      *
第 1 行出现错误:
ORA-06550: 第 1 行, 第 7 列:
PLS-00201: 必须声明标识符 'DBMS_SHARED_POOL.KEEP'
ORA-06550: 第 1 行, 第 7 列:
PL/SQL: Statement ignored
```

显然，提示执行失败，这说明没有可用的软件包，需要手工创建该软件包，如下例所示。该软件包在笔者的电脑上位于目录 F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN 下，脚本文件名为 dbmspool.sql。其实在这个目录下还有很多其他脚本文件，如我们熟悉的、创建 SCOTT 用户的脚本文件 SCOTT.SQL。下面执行 dbmspool.sql 脚本文件，如下例所示。

例子 6-17 创建 DBMS_SHARED_POOL 软件包。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL>@F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN\dbmspool.sql;

程序包已创建。


授权成功。


视图已创建。


程序包体已创建。
```

从输出可以看出，此时成功创建软件包，并且在执行脚本文件 dbmspool.sql 的过程中，实现了授权和视图创建，同时创建了过程 KEEP 和 UNKEEP（当然还有其他过程）。

如果用户在 SCOTT 用户或其他非 SYSTEM 用户下登录数据库，并且尝试创建软件包 DBMS_SHARED_POOL，会提示出错，如下例所示。

例子 6-18 使用非 SYSTEM 用户创建软件包 DBMS_SHARED_POOL。

```
SQL> connect scott/tiger@orcl
已连接。
SQL> F:\app\Administrator\product\11.1.0\db_1\RDBMS\ADMIN\dbmspool.sql;

程序包已创建。


授权成功。

  from dba_object_size
       *
第 4 行出现错误:
ORA-01031: 权限不足


警告: 创建的包体带有编译错误。
```

显然从输出可以看出当前用户缺少足够的权限，只要使用 SYSTEM 用户登录且赋予 DBA 角色即可。

使用 SYSTEM 用户成功创建软件包 DBMS_SHARED_POOL 后，就可以使用它的过程 KEEP 将程序常驻内存了。

6.2.2 将程序常驻内存的过程

软件包 DBMS_SHARED_POOL 是过程的集合，包含常用的 KEEP 和 UNKEEP 过程。使用 KEEP 过程将用户频繁使用的程序常驻共享池中，使用 UNKEEP 将指定的程序从共享池中清除。我们先选择并查看用户 HR 的一个过程。

首先使用 SYSTEM 用户登录数据库。

```
SQL> connect system/oracle@orcl
已连接。
```

然后，通过数据字典 DBA_OBJECTS 查询用户 HR 的一个存储过程，我们假设该存储过程是用户程序频繁调用的过程，然后将其常驻内存。如下例所示，先查找用户 HR 的一个存储过程。

例子 6-19 查看用户 HR 拥有的存储过程。

```
SQL>  select object name,object type
  2   from dba objects
  3   where object type ='PROCEDURE'
  4   and owner ='HR';

OBJECT NAME           OBJECT TYPE
---------------------------------- ------------------------------
SECURE DML            PROCEDURE
ADD_JOB_HISTORY       PROCEDURE
```

从查找结果可以看出，用户 HR 有两个存储过程，一个为 SECURE_DML，另一个为 ADD_JOB_HISTORY。我们将过程 SECURE_DML 常驻内存，或许读者想知道如何查看该过程的内容，毕竟对过程的功能了解得越多就越能理解为什么将其常驻内存，Oralce 提供了数据字典 DBA_SOURCE(USER_SOURCE)。

例子 6-20 查看过程 SECURE_DML 的内容。

```
SQL> SET LINE 100
SQL>select line,text
  2  from dba source
  3* where name ='SECURE_DML'

     LINE   TEXT
---------- ---------------------------------------------------------------
       1 PROCEDURE secure dml
       2 IS
       3 BEGIN
       4  IF TO CHAR (SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00'
       5       OR TO CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
       6    RAISE APPLICATION ERROR (-20205,
       7           'You may only make changes during normal office hours');
       8  END IF;
       9 END secure dml;

已选择 9 行。
```

该过程的作用很简单，就是判断某种状态下的系统时间如果不在 8 点到 18 点之间或者日期为周六或周日就提示错误“You may only make changes during normal office hours”。这里我们不过多分析这个过程，读者只需要知道数据字典 dba_source 的作用即可。下面演示如何将过程 secure_dml 常驻内存，如下例所示。

例子 6-21 将过程 secure_dml 常驻内存。

```
SQL> EXECUTE dbms_shared_pool.keep('HR.SECURE_DML');

PL/SQL 过程已成功完成。
```

输出提示已经成功创建 PL/SQL 过程。为了确认创建结果，我们使用数据字典 v$db_object_cache，它的作用是存储关于数据库对象在缓存中的信息。

例子 6-22 查看用户 HR 的存储过程是否保存在共享池中。

```
SQL> col name for a20
SQL> select name,namespace,sharable mem,executions,kept
  2  from v$db object cache
  3* where owner ='HR'

NAME        NAMESPACE      SHARABLE MEM EXECUTIONS  KEP
----- --------------- ------------- --------- -------- -------
SECURE_DML  TABLE/PROCEDURE    1737         0           YES
```

此时，输出说明用户 HR 的数据库对象即存储过程 SECURE_DML，已经保存在共享池中，因为 KEPT 列的值为 YES。

既然可以使得一个程序常驻内存，同样有方法将其从内存清除。现在我们使用软件包 DBMS_SHARED_POOL 的 UNKEEP 过程，将用户 HR 的过程 SECURE_DML 清除出内存。

例子 6-23 将用户 HR 的过程 SECURE_DML 清除出内存。

```
SQL> EXECUTE dbms_shared_pool.unkeep('HR.SECURE_DML');

PL/SQL 过程已成功完成。
```

在执行清除任务后，我们再次使用数据字典 v$db_object_cache 来查看清除结果，如下例所示。

例子 6-24 查看是否从内存清除过程 SECURE_DML。

```
SQL>  select name,namespace,sharable mem,executions,kept
  2   from v$db object cache
  3   where owner ='HR';

NAME        NAMESPACE       SHARABLE MEM EXECUTIONS KEP
---- ------------------- ---------- -------------- ------
SECURE_DML  TABLE/PROCEDURE  1737            0           NO
```



如果将实现了常驻内存的程序从内存清除，该程序的记录仍然在数据字典 v$db_object_cache 中，只是 KEPT 列的值为 NO。如果从没有将一个程序常驻内存，则在数据字典 v$db_object_cache 中不存在该程序的任何记录。

6.2.3　从 DBMSPOOL 脚本理解软件包 DBMS_SHARED_POOL

上面我们使用软件包 DMBS_SHARED_POOL 将用户 HR 的一个过程 SECURE_DML 常驻内存，同时又使用了软件包过程 UNKEEP 将该过程从内存清除。那么软件包 DBMS_SHARED_POOL 到底是如何创建的呢，我们来分析脚本文件 dbmspool.sql，从而可以更清楚地理解软件包的作用，以及其中包含的过程的含义。以下从脚本文件中截取部分内容详细说明。

```
------------
-- OVERVIEW
--
-- This package provides access to the shared pool.  This is the
-- shared memory area where cursors and PL/SQL objects are stored.
```

这部分说明该软件包的作用是提供对共享池的访问，使得游标或者 PL/SQL 对象（如存储过程、函数等）可以存储在共享池中。

该软件包中还包括四个函数，我们先介绍 KEEP 函数和 UNKEEP 函数。

关于 KEEP 函数，脚本中的内容如下例所示。

```
procedure keep(name varchar2, flag char DEFAULT 'P');
  -- Keep an object in the shared pool.  Once an object has been keeped in
  --   the shared pool, it is not subject to aging out of the pool.  This
  --   may be useful for certain semi-frequently used large objects since
  --   when large objects are brought into the shared pool, a larger
  --   number of other objects (much more than the size of the object
  --   being brought in, may need to be aged out in order to create a
  --   contiguous area large enough.
  --   WARNING:  This procedure may not be supported in the future when
  --   and if automatic mechanisms are implemented to make this
  --   unnecessary.
```

该函数的作用就是将一个数据库对象常驻共享池，使得频繁访问的数据库对象如大对象等减少用户访问的响应时间，提高访问速度。该函数有两个参数。

- 第一个参数 name:
  该参数用来说明数据库对象的名字，这些数据库对象可以是 PL/SQL 过程、触发器、序列号，或者 JAVA 对象。如果是 PL/SQL 过程，可以使用“模式名.过程名”的方式指定特定模式的数据库过程名，如 scott.hispackage。
- 第二个参数 flag:
  该参数的作用是说明要常驻的数据库对象的类型，默认类型为包、过程或函数。否则，需要使用一个字符变量说明对象类型。字符值与其代表的对象类型的对应关系如下例所示。

```
Value        Kind of Object to keep
-----       ----------------------
P            package/procedure/function
 Q           sequence
  R          trigger
  T          type
```

```
    JS          java source
    JC          java class
    JR          java resource
    JD          java shared data
   C            cursor
```

关于 UNKEEP 过程的内容如下例所示。

```
procedure unkeep(name varchar2, flag char DEFAULT 'P');
  -- Unkeep the named object.
  --   WARNING:  This procedure may not be supported in the future when
  --   and if automatic mechanisms are implemented to make this
  --   unnecessary.
  --  Input arguments:
  --   name
  --     The name of the object to unkeep.  See description of the name
  --     object for the 'keep' procedure.
  --flag
  --     See description of the flag parameter for the 'keep' procedure.
  --  Exceptions:
  --   An exception will raised if the named object cannot be found.
```

上面对 KEEP 过程做了详细说明，过程 UNKEEP 的参数含义与 KEEP 过程的相同，所以不再重述 UNKEEP 过程的参数含义。

在该脚本文件中还有一条重要的授权语句，如下例所示。

```
grant execute on dbms_shared_pool to execute_catalog_role
```

将对软件包 DBMS_SHARED_POOL 的执行权利赋予角色 execute_catalog_role，而我们当前的 SYSTEM 用户具有 DBA 权限，所以自动具有角色 execute_catalog_role 的权限。也可以通过数据字典查询当前用户具有的角色，如下例所示。

例子 6-25 查看当前用户的角色信息。

```
SQL> select *
  2  from dba_roles
  3  where role like 'EXECUTE%';

ROLE                          PASSWORD
------------------------- -------------------------
EXECUTE_CATALOG_ROLE             NO
```

上例说明了角色 execute_catalog_role 的存在，所以当前用户在创建了软件包 DBMS_SHARED_POOL 后就可以使用它了。

其实，笔者是希望读者在使用脚本文件时一定要仔细阅读脚本文件的内容，这样就可以从本质上理解一个软件包的作用和其中包含的其他过程。

6.3 将数据常驻内存

在生产数据库中，为了提高用户的访问速度，对于经常使用的表，可以使其常驻内存中，这

样避免了对该表访问时产生频繁的磁盘 I/O 行为，也可以减少用户访问的响应时间。虽然造成一定的内存占用，但是使用内存访问确实减少了访问的响应时间，在某种程度上是有效的。而当不需要频繁访问该表时，DBA 可以将其从内存中清除。本节我们再次学习 Oralce 的各种数据块的缓存池，通过分析了解将数据常驻内存的必要性和可行性。然后给出一个具体的例子将 SCOTT 用户的 EMP 表和一个索引常驻内存。

6.3.1　再论数据块缓存池

在 Oracle 数据库体系结构的介绍中，读者已经知道在数据库块写到磁盘文件之前， 或者从磁盘文件读取数据之后，首先需要将数据块缓存在数据库高速缓存中，所以需要适当设置该缓冲区的大小以满足用户需求。在 Oracle8 之后的版本中，用户可以把 SGA 中段的已缓存块放在 3 个缓冲池中。

- 默认池（default pool）：所有的段都放在这个池中，即原先的缓冲区池。如果没有指定数据的缓存位置，默认将数据缓存在这个池中。
- 保持池（keep pool）：对于用户频繁访问的数据如表或索引等数据库对象的数据块，可以放置在这个候选的缓冲区池中。放在默认池中的数据块，虽然可以频繁访问，但是这些段数据会老化而退出默认池，所以最好放置在保持池中，使得数据可以长久保存。
- 回收池（recycle pool）：对于随机访问的大段可以放在这个缓冲区池中，因为大的数据段会很快老化并退出缓冲池，导致缓冲区的频繁刷新输出，所以需要将随机访问的大段放置这个缓冲区池中。

在 Oracle 数据库中保持池和回收池都是用户管理的，即这两个池的大小需要手工配置，而默认池是自动管理的，在 SGA 中分配。我们通过以下指令可以查看这些保持池的大小信息，如下例所示。

例子 6-26　查看保持池的大小信息。

```
SQL> show parameter keep

NAME                                 TYPE         VALUE
------------------------------------ ------------ -------------
buffer pool keep                      string
control file record keep time        integer        7
db_keep_cache_size                    big integer    0
```

从输出可以看出，对于手动配置的缓冲池保持池的大小，对应的参数 db_keep_cache_size 的值为 0。

在没有设置保持池和回收池前，数据库只使用默认池作为数据块的缓冲池。我们可以通过下例查询当前数据库所使用的数据库块的缓冲池。

例子 6-27　查看当前库的数据块的缓冲池。

```
SQL> select id,name,block_size,buffers
  2  from v$buffer_pool;
```

```
       ID  NAME              BLOCK_SIZE       BUFFERS
    -------------------     ---------------  --------
        3  DEFAULT           8192             47904
```

显然，当前数据库只用一个默认的数据块缓冲池，在手工设置保持池后才会显示保持池的作为数据块缓冲池的信息。

在优化时，我们需要根据实际的需求，将用户经常使用的表或者索引放在保持池中。接下来我们介绍如何设置保持池的大小，以及将数据表和索引常驻内存（保持池）中。

6.3.2 将数据常驻内存的过程

我们将用户 SCOTT 的 SALGRADE 表以及表 EMP 中建立的、基于函数的索引 SCOTT_EMP_INCOME_IDX 常驻保持池中。

通过上节讨论的各种数据块的缓冲池知道，保持池的大小需要手工设置，显然这个尺寸是多少，应该基于常驻保持池中的数据大小，因为我们要将一个表以及索引保存在保持池中，所以需要先确认这些数据库对象的大小，如下例所示。

例子 6-28 查看表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 的块大小。

```
SQL> col segment_name for a20
SQL>select segment_name,segment_type,blocks
  2  from dba_segments
  3  where owner ='SCOTT'
  4* and segment_name in ('SALGRADE','SCOTT_EMP_INCOME_IDX')

SEGMENT_NAME                SEGMENT_TYPE             BLOCKS
-------------------------- --------------------- -----------
SALGRADE                    TABLE                         8
SCOTT_EMP_INCOME_IDX        INDEX                         8
```

表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 的大小都为 8 个数据库块大小。读者需要注意数据字典 dba_segments 是静态数据字典，如果需要获得最新的段统计信息，需要使用 ANALYZE 指令收集统计信息，如下例所示。

例子 6-29 收集表和索引的最新统计信息。

```
SQL> analyze index scott.SCOTT_EMP_INCOME_IDX compute statistics;

索引已分析

SQL> analyze table scott.salgrade compute statistics;

表已分析。
```

在确认了表和索引占用的数据块数后，那么数据库块大小是多少呢？我们通过下例查询库块大小。

例子 6-30 查询的数据库块大小。

```
SQL> show parameter db_block_size;
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------
db_block_size                        integer     8192
```

从输出看出当前数据库的数据块大小为 8K 字节。所以通过这些数据（索引和表的占用的数据块数和数据块大小）可以计算当前表和索引常驻内存需要的内存大小，如下例所示。

例子 6-31 计算要保存的表和索引的大小。

```
SQL> select (8+8)*8 Kbytes from dual;

    KBYTES
----------
       17
```

通过计算得知需要 17KB 的保持池大小，在确认了要保存的数据的大小后，就可以手工设置保持池的大小，如下例所示。

例子 6-32 设置保持池的大小。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL>  alter system set db keep cache size = 10M;

系统已更改。
```

将保持池的大小设置为 10M，这样可以充分满足要存储的包 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 大小的要求。接下来查看当前数据库中数据块的缓冲池信息，如下例所示。

例子 6-33 查询当前库数据块的缓冲池信息。

```
SQL> select id,name,block_size,buffers
  2  from v$buffer_pool;

        ID NAME                 BLOCK_SIZE    BUFFERS
--- --------------- ---------------- ---------------
         1 KEEP                      8192       1497
         3 DEFAULT                   8192      46407
```

从输出可以看出，多了一个缓冲池 keep，该池的数据块大小为 8K，缓冲区大小为 1497 个数据库块大小，即 10M。

下面我们就可以将索引和数据表设置为常驻保持池中了。在设置之前，我们先看看表 SALGRADE 和索引 SCOTT_EMP_INCOME_IDX 当前存放在什么缓冲池中，如下例所示。

例子 6-34 查看表 SALGRADE 当前的存放在什么缓冲池中。

```
SQL> connect scott/tiger@orcl
已连接。
SQL> select table name,tablespace name,buffer pool
  2  from user tables
  3  where table name ='SALGRADE';
```

```
TABLE_NAME          TABLESPACE_NAME     BUFFER_POOL
------------------ ------------------ ----------------------
SALGRADE             USERS               DEFAULT
```

查询结果显示当前的表 SALGRADE 放在默认缓冲池中，因为 BUFFER_POOL 的值为 DEFAULT。下面再查看索引 SCOTT_EMP_INCOME_IDX 的缓存池。

例子 6-35 查看索引 SCOTT_EMP_INCOME_IDX 的缓存池。

```
SQL> select index_name,table_name,buffer_pool
  2  from user_indexes
  3  where index_name ='SCOTT_EMP_INCOME_IDX';

INDEX_NAME                   TABLE_NAME          BUFFER_POOL
------------------------ -------------------- -----------------------
SCOTT_EMP_INCOME_IDX          EMP                 DEFAULT
```

查询结果显示当前的索引 SCOTT_EMP_INCOME_IDX 放在默认缓冲池中，因为 BUFFER_POOL 的值也为 DEFAULT。下面我们将表和索引分别设置为常驻保持池中。

例子 6-36 将表 SALGRADE 设置为常驻内存。

```
SQL> alter table salgrade
  2  storage (buffer_pool keep);

表已更改。
```

现在我们通过数据字典 user_tables 查看表 salgrade 的缓冲池信息，看是否修改为常驻在保持池中，如下例所示。

例子 6-37 查看表 SALGRADE 的缓冲池。

```
SQL> select table_name,tablespace_name,buffer_pool
  2  from user_tables
  3  where table_name ='SALGRADE';

TABLE_NAME          TABLESPACE_NAME    BUFFER_POOL
------------------ ------------------ -------------------------
SALGRADE            USERS               KEEP
```

从列 BUFFER_POOL 的值为 KEEP 可以知道，表 SALGRADE 已经设置为常驻内存（保持池）中了。接下来设置索引常驻内存。

例子 6-38 将索引 scott_emp_income_idx 设置为常驻内存。

```
SQL> alter index scott_emp_income_idx
  2  storage(buffer_pool keep);

索引已更改。
```

现在使用数据字典 USER_INDEXES 查看对索引 SCOTT_EMP_INCOME_IDX 的缓冲池的修改结果，如下例所示。

例子 6-39 查看索引 SCOTT_EMP_INCOME_IDX 的缓冲池。

```
SQL> select index name,table name,buffer pool
  2  from user indexes
  3  where index name = 'SCOTT EMP INCOME IDX';

INDEX NAME                     TABLE NAME             BUFFER POOL
------------------------------ ---------------------- ------------------
SCOTT_EMP_INCOME_IDX             EMP                   KEEP
```

显然，从列 BUFFER_POOL 的值为 KEEP 知道，索引 SCOTT_EMP_INCOME_IDX 已经设置为常驻内存了。

6.3.3 将常驻内存的程序恢复为默认缓冲池

上节我们将用户 SCOTT 的 SALGRADE 表以及表 EMP 中建立的、基于函数的索引 SCOTT_EMP_INCOME_IDX 常驻保持池中。在不需要频繁访问这些表或索引时，可以将其恢复为默认缓冲池，这样就可以释放一部分内存，给其他频繁访问的数据使用。下面先演示如何将表 SALGRADE 恢复为默认缓冲池，如下例所示。

例子 6-40 将表 SALGRADE 恢复为默认缓冲池。

```
SQL> alter table salgrade
  2  storage(buffer_pool default);

表已更改。
```

接着可以查看修改结果，确认是否将表 SALGRADE 的缓冲池设置为默认缓冲池，如下例所示。

例子 6-41 查看表 SALGRADE 的缓冲池信息。

```
SQL> select table name,tablespace name,buffer pool
  2  from user tables
  3  where table_name ='SALGRADE';

TABLE NAME           TABLESPACE NAME      BUFFER POOL
-------------------- -------------------- ---------------------
SALGRADE              USERS                 DEFAULT
```

输出说明已经将表 SALGRADE 常驻内存改为使用默认缓冲区，因为 BUFFER_POOL 的值已经为 DEFAULT，以后对表 SALGRADE 的访问将把表数据读入默认缓冲区。

接下来将索引 SCOTT_EMP_INCOME_IDX 从常驻内存改为使用默认缓冲池，如下例所示。

例子 6-42 将索引 SCOTT_EMP_INCOME_IDX 恢复为默认缓冲池。

```
SQL> alter index scott emp income idx
  2  storage (buffer pool default);

索引已更改。
```

其实，与设置为常驻内存不同的是，STORAGE 子句中的一个参数，将设置常驻内存的 KEEP 参数改为 DEFAULT 就修改了索引的缓冲池设置。此时，我们成功将索引 SCOTT_EMP_

INCOME_IDX 的缓冲池设置为默认缓冲池。

显然此时，保持池依然占用内存，但是其中已经没有了数据，那么如何释放保持池中的内存呢？我们使用 ALTER SYSTEM 指令来回收这段内存，如下例所示。

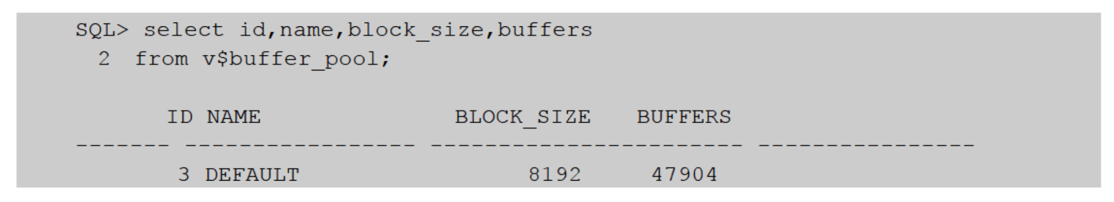
例子 6-43 会收保持池中的内存。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> alter system set db keep cache size = 0;

系统已更改。
```

此时，我们不再使用保持池作为缓冲池，可以使用数据字典 v$buffer_pool 来验证。

例子 6-44 查看与数据库相关的缓冲池信息。



```
SQL> select id,name,block_size,buffers
  2  from v$buffer_pool;

      ID NAME                BLOCK_SIZE    BUFFERS
-------- ------------------ -------------------------- -----------------
       3 DEFAULT                   8192      47904
```

显然此时只有默认缓冲池可以使用，说明保持池已经不再有效。

6.4 优化重做日志缓冲区

重做日志缓冲区是一段临时存储重做数据的内存区，用户所有的修改前数据和修改后的数据都保存在重做日志缓冲区中，由 LGWR 进程负责写入重做日志文件。在优化时，需要考虑该内存区的大小，以及 LGWR 的写速度和重做日志文件所在磁盘的争用等。本节我们首先重述重做日志文件的工作机制，理解与重做日志相关的等待事件，最后给出相应地解决问题的思路，达到优化重做日志缓冲区的目的。

6.4.1 深入理解重做日志缓冲区的工作机制

在 SGA 中重做日志缓冲区一般是最小的一个内存结构。在用户对数据库做更改时，重做日志缓冲区为所有修改数据的服务器进程共享使用。这些服务器进程负责将更改数据的原始值和修改后的新值以及事务 ID 写入重做日志缓冲区，而 LGWR 进程负责将重做日志缓冲区中的数据写入重做日志文件。Oracle 的重做日志组是循环使用的，当覆盖以前的重做日志文件时，如果数据库处于归档模式，则自动启动归档进程。ARCH 负责将被覆盖的重做日志文件的内容拷贝到归档日志文件，图 6-2 是以上描述的行为的示意图。

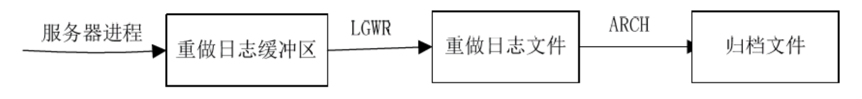


图 6-2 重做日志缓冲区以及相关进程的工作示意图

我们看到上图是一个静态图，其实在数据库内部上述活动是个十分活跃的行为。服务器进程修改数据库中的数据或表结构，不断地将相关的重做数据写入重做日志缓冲区，而重做日志缓冲区在一定的条件下，比如每 3 秒钟，将其中的重做数据写入重做日志文件。而当重做日志文件切换时（无论是用户主动切换，还是数据库自己的行为）导致归档进程 ARCH 启动，把重做日志文件中的数据读入归档文件，然后数据库才可以继续使用相关的重做日志文件。在这个过程中无论哪里出现问题都会导致一些等待事件，如果是频繁发生的等待事件，就会影响系统的性能，比如重做日志缓冲区太小，而服务器进程写入速度又比 LGWR 写出的速度快，就会出现 log buffer space 等待时间，此时就需要 DBA 主动采取优化了。

为了更清楚的理解在用户修改一行数据库时，与重做日志相关的一系列行为，我们给出一个过程分析。

01 用户发出一条更新的 SQL 语句，该语句是某个事务的一部分，Oracle 为该事务分配了唯一的事务号。

02 服务器进程负责将需要的数据、索引和还原数据读入内存，并将要更新的行加锁。

03 服务器进程获得重做拷贝闩锁（闩锁实现对重做日志缓冲区的串行使用），该闩锁是服务器进程访问重做日志缓冲区的第一步。此时如果没有其他的闩锁可用，则别的服务器进程无法使用重做日志缓冲区。

04 服务器进程获得重做分配闩锁从而获得在重做日志缓冲区中的预留空间，此时释放重做分配闩锁。

05 服务器进程利用重做拷贝闩锁把重做项（更新数据的原始值，操作类型、事务号等信息）写入重做日志缓冲区。然后释放重做日志拷贝闩锁。

06 服务器进程把还原信息写入与该事务相关的还原段，还原段在用户使用 ROLLBACK 指令时使用。

07 服务器进程更新锁住的数据，将回滚所需的原始值和对数据所做的修改都写入数据库高速缓冲区，然后数据库高速缓冲区中的这些数据被标记为脏数据，因为目前内存和外存中的数据不一致。

在深入地了解了重做日志缓冲区的工作机制和过程后，我们分析 LGWR 进程何时将重做日志缓冲区的重做数据写入重做日志文件，理解这些内容对于优化重做日志缓冲区是很必要的。以下列出 LGWR 把重做日志缓冲区写入重做日志文件的条件。

- 每隔 3 秒钟。
- 事务被提交时。
- 当重做日志缓冲区的记录的变化的数据量超过 1M 字节。
- 当重做数据的大小为重做日志缓冲区大小的 1/3 时。这里需要说明，并不是重做日志缓冲区永远不会填到超过其 1/3 容量，而是说明当重做数据量达到其容量的 1/3 这个阈值时，LGWR 进程会写出重做日志缓冲区中的数据，而剩下的 2/3 的数据可以供其他服务器进程使用。
- 检验点发生时。
- 当 DBWR 进程将数据库高速缓冲区中的数据写到数据文件前。

下面我们查看重做日志缓冲区的尺寸，如下例所示。

例子 6-45 查看重做日志缓冲区的尺寸。

```
SQL> show parameter log_buffer;

NAME                                 TYPE             VALUE
------------------------------------ ---------------- ----------
log_buffer                           integer          7024640
```

6.4.2 重做日志缓冲区相关的等待事件

如果需要优化重做日志缓冲区，必须首先确认发生了与重做日志缓冲区相关的等待事件，否则不应该随便调整重做日志缓冲区的尺寸。读者可以通过相关等待（WAIT）视图和事件(EVENT)视图，确认等待事件以及该事件涉及的文件和会话，如下例所示。

例子 6-46 通过数据字典视图查看会话等待事件。

```
SQL> col event for a35
SQL> col username for a10
SQL> select sw.sid,s.username,sw.event,sw.wait_time
  2  from v$session s,v$session wait sw
  3  where sw.event not like 'rdbms%'
  4  and sw.sid = s.sid
  5* order by sw.wait time,sw.event

       SID USERNAME   EVENT                                WAIT_TIME
---------- ---------- ----------------------------------- ------------
       154 SYSTEM     SQL*Net message to client                   -1
       149            Streams AQ: qmn coordinator idle wa          0
                      it

       144            Streams AQ: qmn slave idle wait              0
       147            Streams AQ: waiting for time manage          0
                      ment or cleanup tasks

       159            jobq slave wait                              0
       170            pmon timer                                   0
       164            smon timer                                   0

已选择 7 行。
```



虽然出现了等待事件，但是该等待事件没有影响系统使用或系统性能，就不要轻易去优化。

下面我们分析和重做日志缓冲区相关的等待事件，以及事件发生的原因。一旦找到等待事件，并知道该事件发生的相关原因，就可以实现优化工作了。

- **Log buffer space**：该事件说明缺少重做日志的缓冲区空间，造成该等待事件的原因一般是服务器进程写入重做日志缓冲区的速度高于 LGWR 将重做日志缓冲区写出的速度。也有

可能是重做日志文件所在磁盘设备速度慢或者存在设备争用，造成 LGWR 进程无法及时将重做日志缓冲区中的重做数据写入重做日志文件。

优化方法：调整重做日志缓冲区尺寸，或者将重做日志数据文件迁移到高速磁盘上，或者为了解决争用，将重做日志文件和数据库数据文件以及归档文件放在不同的磁盘上。

- **Log file parallel write**：该事件的含义是日志文件并行写等待，是在将重做日志缓冲区中的重做数据写入磁盘引起的等待事件。造成该事件的原因一般是联机重做日志文件所在的设备速度慢或者存在磁盘争用。

  优化方法：将重做日志文件和数据库数据文件以及归档文件放在不同的磁盘上。或者将重做日志文件放置在高速盘上。

- **Log file single write**：该等待事件仅与写日志文件头块有关，表示检查点中的等待。
- **Log file switch(archiving needed)**：该等待事件的含义是日志文件切换等待。对于处于归档模式的数据库而言，当日志组写满后，在日志切换时，如果需要覆盖先前的日志，而该日志需要归档进程写入归档文件，由于写入归档文件需要时间，而 LGWR 进程需要将重做日志缓冲区中的数据写入重做日志文件，而归档未完成需要等待，在此期间就产生了 Log file switch 事件。该等待事件的原因一般是 I/O 问题、ARCH 归档进程跟不上 LGWR 日志写进程的速度或者日志组太少引起的。

  优化方法：启用多个归档 ARCH 进程或 I/O 从进程（slave process），将归档的文件和数据文件或重做日志文件放置在不同的磁盘上，减少磁盘争用以减少 ARCH 归档进程的归档事件，或者增加重做日志组。

- **Log file switch(checkpoint incomplete)**：该事件由于日志切换太频繁引起的。频繁地切换重做日志文件，造成检验点的排队。发生该等待事件的原因一般是重做日志缓冲区空间太小或者重做日志组太少。

  优化方法：增加重做日志组或者增加重做日志缓冲区的尺寸。

- **Log file sync**：当用户提交时，重做日志缓冲区中的数据会一次全部写到重做日志文件中，此时发生的 LGWR 的写出等待就是 log file sync 等待。造成该等待原因一般是放置联机重做日志文件的磁盘存在争用或者磁盘速度慢。

  优化方法：将重做日志文件和数据文件或归档重做日志文件放置在不同的磁盘上，以减少数据库中的各种文件之间的 I/O 争用，同时可以把重做日志文件放在高速磁盘上，以减少将重做数据写入重做日志文件的时间。

- **Latch free**：该等待事件的含义是当前的服务器进程需要某个闩锁，比如等待共享池的库高速缓存闩锁。如果发生该等待事件，也可以通过数据字典 v$latch 查看相关的闩锁命中率，如下例所示。

例子 6-47 查询与闩锁 LATCH 相关的信息。

```
SQL> select latch#,name,gets,misses,1-(misses/(gets+misses)) "gets rate"
  2  from v$latch
  3  where misses>1;

    LATCH# NAME                              GETS     MISSES    gets rate
-------- ------------------------- ---------- ---------- ----------
      180 dummy allocation                   1605       2        .998755445
```

```
      121 checkpoint queue latch         24231     9         .99967713
      213 shared pool                    96480     112       .998840484
    .................
      215 library cache lock             45617     3         .999934239
      199 row cache objects              80118     9         .999887678
      214 library cache                  135162    59        .999563677
      146 redo writing                   6541      115       .982722356
      216 library cache pin              7765      3         .99995883
      19 enqueue hash chains             69022     5         .999927565
      302 session state list latch       1602      5         .996888612

已选择 18 行。
```

可以通过以上所示的 get rate 字段的值来进一步确认闩锁的命中率。

在上述与重做日志缓冲区有关的等待事件的分析中，我们详细分析了造成这种等待事件的原因，读者或许可以体会到，理解重做日志缓冲区的工作机制，以及它涉及的各种进程对于理解这些等待事件是很有好处的。我们知道一个事件的含义，又可以分析事件发生的原因，对于解决这个等待事件也就有了相应的解决思路。其实我们已经给出了解决等待事件的思路和优化方法，只是在具体操作时，读者需要使用以前章节中介绍的方法，如迁移数据文件、创建重做日志组以及向重做日志组中添加日志文件等。

6.4.3 设置重做日志缓冲区大小

在发生与重做日志缓冲区相关的等待事件时，或者在 DML 操作频繁的生产数据库中，往往需要增加重做日志缓冲区的尺寸。下面我们介绍如何修改重做日志缓冲区的大小并使其生效。先查看重做日志缓冲区的大小，如下例所示。

例子 6-48 查看重做日志缓冲区的大小。

```
SQL> show parameter log_buffer;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
log_buffer                           integer     7024640
```

从输出可以看出该重做日志缓冲区的大小为 7024640 字节，所以在设置该参数的值时，也必须用字节数来修改该参数。为了更容易理解，我们使用将字节数据转换成 M 字节的形式，并使用数据字典视图 v$parater，如下例所示。

例子 6-49 使用数据字典视图 v$parameter 查看重做缓冲区的大小。

```
SQL> col name for a20
SQL> select name,value/(1024*1024) "M 字节"
  2  from v$parameter
  3* where name ='log buffer'

NAME                 M 字节
-------------------- ----------
log_buffer           6.69921875
```

可以看到当前数据库的重做日志缓冲区的尺寸为 7M。由于参数 LOG_BUFFER 是静态参数，所以设置该参数后必须重新启动数据库才可以生效。下面我们增大重做日志缓冲区的尺寸，设置为 10M。我们先将 10M 转换成字节（10*1024*1024=10485760 字节），如下例所示。

例子 6-50　设置重做日志缓冲区大小为 10M。

```
SQL> alter system set log buffer = 10485760 scope = spfile;

系统已更改。
```

为了使设置的重做日志缓冲区参数生效，必须关闭数据库重新启动。下面我们关闭数据库并重新启动数据库。

例子 6-51　关闭并重启数据库。

```
SQL> connect system/oracle@orcl as sysdba
已连接。
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  734003200 bytes
Fixed Size                  171172 bytes
Variable Size             197134492 bytes
Database Buffers           52478000 bytes
Redo Buffers               11329536 bytes
数据库装载完毕。
数据库已经打开。
```

重新打开数据库后，修改过的重做日志缓冲区的大小生效。我们查询一下重做日志缓冲区的修改结果，如下例所示。

例子 6-52　查询重做日志缓冲区的尺寸。

```
SQL> show parameter log_buffer;

NAME                                 TYPE          VALUE
------------------------------------ ------------- ------------
log_buffer                           integer       11154432
```

从结果中可以看到此时 LOG_BUFFER 的 VALUE 为 11154432 字节，已经不是以前的 7024640 字节，说明在例子 6-50 中修改生效了。



这里设置的该缓冲区大小为 10M，但是其数值为 10.6376953M，是因为 Oracle 会根据系统情况做一些调整。

6.5 优化共享池（Shared Pool）

在 Oracle 数据库系统架构中，共享池由两部分组成：库高速缓存和数据字典高速缓存。其中库高速缓存存放 SQL 语句的正文、编译后的代码以及最终的执行计划，而数据字典高速缓存存放 SQL 语句操作相关的数据库对象，如表、索引、列以及其他对象的定义和权限信息。

对于库高速缓存而言，重用 SQL 语句可以减少硬解析的时间从而减少 SQL 语句的响应时间。而数据字典高速缓存则减少了对 SQL 语句涉及的数据库对象和权限定义的磁盘访问，也减少了 SQL 语句的响应时间。对共享池优化目的就是在不影响性能的条件下提高 SQL 代码以及数据字典的使用率。

6.5.1 库高速缓存

库高速缓存存放 SQL 语句的正文、编译后的代码以及最终的执行计划。而在 SQL 语句的处理步骤中，对 SQL 语句的解析是最耗费时间的，解析需要经过 SQL 语句的语法语义分析、基于优化模式选择优化方案、执行最终执行计划，这种解析称为硬解析。如果有相同的 SQL 语句的执行计划已经缓存在库高速缓冲区中，此时就只执行一个软解析，软解析通过 SQL 语句的正文比对找到该语句的最终执行计划。所以，要尽量减少硬解析的发生。

6.5.2 使用绑定变量

Oracle 如何判断两个 SQL 语句是相同的呢？Oracle 对语句相同的判断条件很苛刻，要求两个语句的正文必须一样，这包括空格以及字母的大小写情况。下例所示的两个语句 Oracle 认为是不同的。

SQL 查询语句一：

```
SQL> select ename,sal,mgr
  2  from scott.emp;
```

SQL 查询语句二：

```
SQL> select ename,sal,MGR
  2  from scott.emp;
```

二者的区别在于列 MGR 的大小写不同，虽然二者的查询结果相同，但是 Oracle 认为这两个语句是不同的，对于第二个语句会发生硬解析，大量硬解析会造成内存、CPU 以及闩的争用。

在具有如下相似的查询时，读者最好使用绑定变量。使用绑定变量，Oracle 认为使用相同条件但是不同参数值的 SQL 语句是相同的，这样可以减少硬解析的发生，如下例所示。

```
SQL> select ename,sal,mgr from scott.emp where sal>5000;
SQL> select ename,sal,mgr from scott.emp where sal>2000;
SQL> select ename,sal,mgr from scott.emp where sal>3000;
```

对于上述的查询，Oracle 认为这是三个独立的查询，因为三个语句的正文并不完全相同，所以，此时这样的查询最好使用绑定变量，如下例所示。

例子 6-53 使用绑定变量

```
SQL> select ename,sal,mgr from scott.emp where sal>&salary
输入 salary 的值:  5000
原值    1: select ename,sal,mgr from scott.emp where sal>&salary
新值    1: select ename,sal,mgr from scott.emp where sal>5000

未选定行

SQL> select ename,sal,mgr from scott.emp where sal>&salary
输入 salary 的值:  2000
原值    1: select ename,sal,mgr from scott.emp where sal>&salary
新值    1: select ename,sal,mgr from scott.emp where sal>2000

ENAME            SAL       MGR
---------- ---------- ----------
JONES           2975       6839
BLAKE           750       7839
CLARK           2450       7839
SCOTT           3000       7566
KING            5000
FORD            3000       7566

已选择 6 行。
```

上例通过绑定变量&salary 来代替具体的值，在查询时刻再输入具体的值，这样 Oracle 就认为使用相同的 SQL 语句，即 select ename,sal,mgr from scott.emp where sal>&salary，这样就极大的减少了硬解析的数量。

上面已经提过了硬解析和软解析的概念，这里再详细分析并给出一个示例说明，硬解析的过程包括 SQL 语句的语义和语法分析、检查对象和用户权限等信息，基于优化方式生成最终执行计划，最后将执行计划保存在库高速缓存，显然整个过程占用大量的 CPU 时间，并会引起闩争用（因为此时的库高速缓存被占用），势必会增加 SQL 语句的响应时间。

软解析则没有 SQL 语句的语法和语义分析以及基于优化方式生成执行计划的过程，比对 SQL 语句的正文（通过对语句的散列实现），发现库高速缓存中的相同 SQL 语句，执行该语句的执行计划。显然软解析减少了 CPU 的计算时间，也减少了闩的争用。下面我们通过 SQL Trace 跟踪一个 SQL 查询语句，查看是否发生硬解析，具体步骤如下所示。

01 清空共享池（目的是方便对 SQL 语句的分析）。

```
SQL> alter system flush shared pool;

系统已更改。
```

02 启动会话级的 SQL 追踪功能。

```
SQL> alter session set sql trace=true;

会话已更改。
```

03 执行 SQL 查询语句并通过 TKPROF 解释该追踪文件。

例子 6-54 执行 SQL 查询语句

```
SQL> select ename,sal,mgr from scott.emp where sal>4000;

ENAME              SAL        MGR
---------- ---------- ----------
KING              5000
```

在开始时将共享池清空，这样执行的 SQL 查询语句必须被硬解析。我们打开会话追踪文件，发现上述语句的解析过程，如下内容所示。

```
PARSING IN CURSOR #3 len=50 dep=0 uid=0 oct=3 lid=0 tim=5429793646 hv=68121321
ad='22d049b8'
select ename,sal,mgr from scott.emp where sal>4000
END OF STMT
PARSE
#3:c=187500,e=304194,p=0,cr=176,cu=161,mis=1,r=0,dep=0,og=2,tim=5429793642
EXEC #3:c=0,e=19,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=5429793715
FETCH #3:c=0,e=46,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=5429793808
FETCH #3:c=0,e=9,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=5429794112
STAT #3 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
 time=39 us)'
```

在上述追踪文件中，参数 mis 对应的值说明是否发生硬解析，如果该参数值为 1 说明发生了硬解析，因为此时的库高速缓存丢失一次命中。

下面我们再执行一个类似的查询，看是否发生了硬解析，从而使得读者对使用绑定变量有更深入的理解，如下例所示。

例子 6-55 执行一个与例子 7-51 类似的查询。

```
SQL> select ename,sal,mgr from scott.emp where sal>3000;

ENAME              SAL        MGR
---------- ---------- ----------
KING              5000
```

此时，再查看追踪文件，相关内容如下例所示。

```
PARSING IN CURSOR #2 len=50 dep=0 uid=0 oct=3 lid=0 tim=6111445032 hv=696689503
ad='22d4677'
select ename,sal,mgr from scott.emp where sal>3000
END OF STMT
PARSE
#2:c=15625,e=26456,p=0,cr=19,cu=26,mis=1,r=0,dep=0,og=2,tim=6111445023
EXEC #2:c=0,e=45,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=6111445200
FETCH #2:c=0,e=96,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=6111445387
FETCH #2:c=0,e=20,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=6111446269
STAT #2 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
time=83 us)'
```

从输出可以看出此时的 mis 值为 1，说明此时使用了硬解析，因为两个 SQL 语句的正文并不

相同。下面，再一次输入 select ename,sal,mgr from scott.emp where sal>3000 语句，继续跟踪文件，如下例所示。

```
    PARSING IN CURSOR #2 len=50 dep=0 uid=0 oct=3 lid=0 tim=8689398915 hv=696689503
    ad='22da97d8'
    select ename,sal,mgr from scott.emp where sal>3000
    END OF STMT
    PARSE #2:c=0,e=125,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=8689398906
    EXEC #2:c=0,e=57,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=2,tim=8689399144
    FETCH #2:c=0,e=132,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=2,tim=8689399361
    FETCH #2:c=0,e=19,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=2,tim=8689399790
    STAT #2 id=1 cnt=1 pid=0 pos=1 obj=51151 op='TABLE ACCESS FULL EMP (cr=8 pr=0
pw=0
    time=119 us)'
```

从跟踪文件的输出看出，参数 mis=0 说明此时没有硬解析。

6.5.3　调整参数 CURSOR_SHARING 参数

该参数默认值为 EXACT，意思是只有正文完全相同的 SQL 语句才可以重用，该值也是参数 CURSOR_SHARING 的默认值，如下例所示。

例子 6-56　查看参数 CURSOR_SHARING 的默认值。

```
    SQL> show parameter cursor sharing;

    NAME                                TYPE         VALUE
    ------------------------------- ----------- ---------
    cursor_sharing                      string       EXACT
```

如果缺少绑定变量，而系统中有大量只有字面值不同的 SQL 语句，此时可以通过更改参数 CURSOR_SHARING 的值为 FORCE 来缓解 SQL 语句的硬解析开销。如果将参数 CURSOR_SHARING 的参数为 FORCE，则强制共享使用只有字符不同的 SQL 语句。如下例所示，我们修改这个参数值。

例子 6-57　将参数 CURSOR_SHARING 的参数为 FORCE。

```
    SQL> alter system set cursor sharing=force scope=both;

    系统已更改。
```

6.5.4　设置共享池的大小

在 Oracle 10g 及以上版本中，共享池的大小由 SGA 自动调整。只要设置了 SGA_TARGET 参数，其他 SGA 组件的大小都有 Oracle 自己调整。如果是自动的 SGA 调整，我们查询一下共享池的大小，如下例所示。

例子 6-58　查询共享池的大小。

```
    SQL> show parameter shared_pool_size;
```

```
NAME                          TYPE              VALUE
----------------------------- ----------------- ------------
shared_pool_size              big integer        0
```

此时输出参数 shared_pool_size 的值为 0，因为此时是自动的 SGA 管理，所以 Oracle 不会显示该参数的值，在需要的时候会从 SGA 自动分配。

在安装数据库时，我们可以选择自动 SGA 管理，此时只需要告诉 SGA 占有操作系统内存的比例，或设置相应的值，如图 6-3 所示。

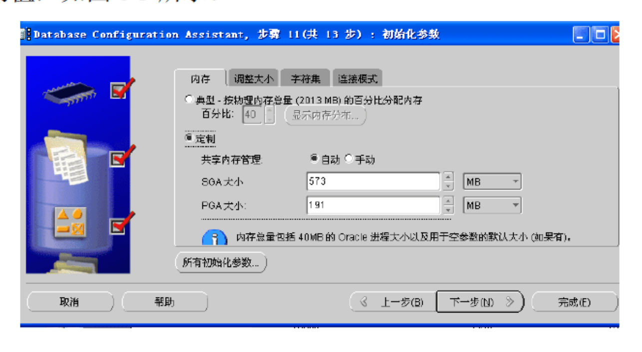


图 6-3　设置 SGA 自动管理

在安装数据库时也可以手工设置 SGA 各内存组件的大小，如图 6-4 所示。



图 6-4　手工设置 SGA 组件大小

对于手工设置 SGA 大小的方式，在数据库运行期间也可以使用 ALTER SYSTEM SET 指令修改共享池的大小，如下例所示。

例子 6-59　手工设置共享池的大小。

```
SQL> alter system set shared_pool_size=200M scope =both;

系统已更改。
```



在 SGA 自动调整的条件下，也可以使用上述指令手工设置共享池的大小。

修改后，再查询一下修改结果，验证修改是否成功，如下例所示。

例子 6-60　查询共享池大小是否修改成功。

```
SQL> show parameter shared_pool_size;

NAME                          TYPE                   VALUE
------------------------ --------------------- -----------------
shared_pool_size             big integer            200M
```

6.6　优化数据库高速缓存（DB Cache）

数据库高速缓存中存储了最近从数据文件读入的数据块信息或用户更改后需要写回数据库的数据信息，此时这些没有提交给数据库的更改后的数据称为脏数据。当用户执行查询语句如 select * from dept 时，如果用户查询的数据块在数据库高速缓存中，Oracle 就不必从磁盘读取，而是直接从数据库高速缓存中读取，显然从内存读取的速度要快很多，这些缓存的数据由 LRU 算法管理。通过数据库高速缓存提高了用户的查询速度，减少用户查询的响应时间。Oracle 使用 LRU 算法管理库缓冲区，把最近没被使用的数据块从库高速缓冲中删除，为其他的查询数据块保留空间。所以应该尽量把数据库高速缓存设置大些，这样就能为用户提供更多的可查询缓存数据，但是也要注意 SGA 的大小是受限的，毕竟还有其他内存组件需要空间，过大的数据库高速缓存也是不可取的。本节我们讨论如何优化数据库高速缓存。

6.6.1　调整数据库缓冲区大小

数据库缓冲区存储用户读取或要修改的数据，设置合适的缓冲区大小可以提高缓冲区的命中率，提高用户数据的响应时间。计算数据库缓冲区命中率之前需要知道的几个缓冲区值，如下例所示。

例子 6-61　查询相关缓冲区缓存数据。

```
SQL> select name,value
  2  from v$sysstat
  3  where name in ('db block gets from cache',
  4  'consistent gets from cache',
  5  'physical reads cache');

NAME                                                VALUE
------------------------------------------------ ----------
db block gets from cache                            47486
consistent gets from cache                          329934
physical reads cache                                9813
```

我们使用数据字典视图 v$sysstat 查询了三个与计算数据库缓冲区命中率相关的缓冲区缓存数据，下面详细分析一下与这三个缓存区相关的三个术语。

- db block gets：DB 块获取。在数据库高速缓冲区中，如果存在被更改的数据，而此数据在其他用户访问时已经提交，也就是用户访问的数据在数据库缓冲区中是最新的版本，这样的数据块读称为 db block gets，即 DB 块获取。
- consistent gets：一致获取。在数据库高速缓冲区中，如果存在被更改的数据，而此数据在

其他用户访问时还没有提交，也就是用户访问的数据在数据库缓冲区中是脏数据，这样的数据不会被访问，此时用户只能使用回滚段中的记录，这样的数据块读称为 consistent gets，即一致获取。

- physical reads：物理读。在数据库高速缓冲区中，没有用户要访问的数据，需要从磁盘读取该数据块，这样的数据读取称为 physical reads，即物理读。

DB 块获取和一致获取都是逻辑读，在计算数据库高速缓冲区的命中率时，需要使用物理读和逻辑读的比值作为参考，即：

命中率=1-物理读/逻辑读。

逻辑读= db block gets from cache + consistent gets from cache= 377420。

物理读= 9813。

所以此时数据库高速缓冲区命中率为：

```
    p = 1-physical reads cache/( db block gets from cache + consistent gets from
cache)
    = 1-9813/377420=0.975669016.
```

显然这样的库缓冲区的命中率是不错的。



命中率并不能说明此时的数据库性能就一定很好，还需要具体分析等待事件。如果此时的命中率很高，但是系统存在大量的等待事件，如数据文件离散度等，可能存在大量的全表扫描，就需要进一步分析等待事件和相关的 SQL 语句。

如果对自己的应用环境非常熟悉，可以采用手工设置数据库高速缓冲区的大小，此时在数据库系统的典型运行阶段查询命中率，反复调整，直到命中率满足要求。修改库高速缓冲区大小，如下例所示。

例子 6-62 修改库高速缓冲区大小。

```
    SQL> alter system set db_cache_size=192M;

    系统已更改。
```

6.6.2 使用缓冲池

如果用户访问使用了多个表，并且这些表都相当大，此时这些表只有部分保存在库高速缓冲区中，库缓冲区中的大表数据的存在会降低数据库缓冲区的命中率，而如果这样的大表不是用户频繁访问的对象，就可以使用回收缓冲池（recycle pool）来存放，这样的数据库对象会被覆盖。

如果用户访问对多个小表全表扫描，且这样的行为频繁发生，就可以将这些小表保存在保留池（keep pool）中，这样的数据库对象不会被覆盖掉。

对于其他既没有存储在回收缓冲池，也没有存储在保留池中的数据库对象，默认存储在数据库缓冲区的默认池（default pool）中。



缓存表的尺寸不要超过缓冲区大小的 10%。

下面是将数据对象存储在缓冲池中的语法。

```
SQL> create index t idx Storage (buffer pool keep);
SQL> alter table t name storage(buffer pool recycle);
SQL> alter index t_idx  storage(buffer_pool keep);
```

接下来我们给出一个例子说明如何在创建一个索引的同时，将该索引调入保留池，如下例所示。

例子 6-63　创建索引并将其缓存在保留池。

```
SQL> create index scott_emp_ename
  2  on scott.emp(ename)
  3  storage (buffer_pool keep);

索引已创建。
```

为了验证保存结果，我们通过数据字典 user_indexes 视图查看 buffer_pool 列的值，如果该值为 KEEP，则说明索引已经保存在保留池中。

例子 6-64　查询索引 scott_emp_ename 是否缓存在保留池中。

```
SQL> col buffer pool for a15
SQL> run
  1  select index_name,table_name,buffer_pool
  2  from user indexes
  3* where index_name='SCOTT_EMP_ENAME'

INDEX NAME                     TABLE NAME                     BUFFER POOL
------------------------------ ------------------------------ ---------------
SCOTT_EMP_ENAME                EMP                             KEEP
```

从输出看出，索引 SCOTT_EMP_ENAME 已经缓存在保留池中，因为列 BUFFER_POOL 的值为 KEEP。

如果是表，则可以通过数据字典 user_tables 来查看，该视图的 cache 列说明该表是否被缓存，buffer_pool 列说明该表缓存的缓冲池名称。下面我们将表 EMP 放入回收池，如下例所示。

例子 6-65　将表 EMP 缓存在回收池。

```
SQL> alter table scott.emp
  2  storage (buffer_pool recycle);

表已更改。
```

此时，我们将 SCOTT 用户的 EMP 表存储在回收池中。下面通过数据字典 dba_tables 查看更改结果，如下例所示。

例子 6-66　查看表 EMP 的缓存信息。

```
SQL> select table_name,cache,buffer_pool
  2  from dba_tables
  3  where table name='EMP';

TABLE_NAME                     CACHE      BUFFER_POOL
------------------------------ ---------- ----------------------------
```

```
EMP                                Y          RECYCLE
```

列 CACHE 说明表 EMP 已经缓存在缓冲区中，而列 BUFFER_POOL 说明对应的缓冲区为回收池 RECYCLE POOL。

我们也可以使用 CACHE 关键字，在创建一个表时，说明如果该表被缓存则缓存在默认池中，如下例所示。

例子 6-67 使用 CACHE 关键字将表缓存在默认池中。

```
SQL> create table test
  2  as
  3  select *
  4  from scott.dept
  5  cache;

表已创建。
```

下面，通过数据字典 USER_TABLES 查看表 TEST 的缓存结果，如下例所示。

例子 6-68 查看表 TEST 的缓存信息。

```
SQL> select table_name,cache,buffer_pool
  2  from user_tables
  3  where table_name='TEST';

TABLE_NAME                     CACHE        BUFFER_POOL
------------------------------ ---------- ---------------
TEST                             N            DEFAULT
```

此时，我们发现表 TEST 会使用 DEFAULT 缓冲池，但是目前没有被缓存，所以，还需要使用指令启动表的缓存，如下例所示。

例子 6-69 将表 TEST 缓存在默认池中。

```
SQL> alter table test cache ;

表已更改。
```

我们继续通过数据字典 USER_TABLES 查看表 TEST 的缓存结果，如下例所示。

例子 6-70 查看表 TEST 的缓存信息。

```
SQL> select table_name,cache,buffer_pool
  2  from user_tables
  3  where table_name='TEST';

TABLE_NAME                      CACHE        BUFFER_POOL
------------------------------ ------------ ---------------
TEST                              Y            DEFAULT
```

此时列 CACHE 的值为 Y，说明已经将该表加载到缓冲区，此缓冲区为 DEFAULT 缓冲区。

6.7　优化PGA内存

PGA 是程序全局区，该区域在数据库会话专有连接模式下是私有区域，服务器进程和用户进程一一对应，用户进程单独使用 PGA，在共享服务器会话连接模式下，一个服务器进程对应多个用户进程，PGA 是多个用户进程共享使用。PGA 主要用于大规模的数据排序，如用户输入的 SQL 语句有 GROUP BY 或 ORDER BY 子句等操作。

所谓的 PGA 优化就是将这些大规模的数据排序放在 PGA 中运行，避免使用虚拟内存而占用操作系统的交换区（SWAP AREA）。这样就要求合理地设置 PGA 的大小，从而使得排序区符合系统需要，在 Oracle 9i 之前，该排序区由参数 SORT_AREA_SIZE 决定，通过手工设置该参数，反复调整以满足系统需要，可以通过如下的方式查看该参数的值。

例子 6-71　查看 PGA 中排序区的大小。

```
NAME                               TYPE         VALUE
---------------------------------- ------------ ----------
sort_area_size                     integer      65536
```

如果出于大规模排序的需要，可以调整该参数为更大的值，但是问题是到底多少是最好的？显然如果分配过多，则造成系统的其他组件的内存不足，如果过少又不能满足排序的需要。在 Oracle 9i 及以上版本中（Oracle 10g、Oracle 11g）支持 PGA 排序区的自动调整功能。但是该自动调整有一个限制，就是必须给出一个排序区的值，排序区会根据排序需要在这个值内自动调整。该值由参数 PGA_AGGREGATE_TARGET 决定。同时为了启动 PGA 排序区的自动管理，必须设置参数 WORKAREA_SIZE_POLICY 为 AUTO。这也就是说设置 PGA 排序区的自动管理必须配置两个参数，即 PGA_AGGREGATE_TARGET 和 WORKAREA_SIZE_POLICY。下面我们查看系统上这两个参数的值，如下例所示。

例子 6-72　查看 PGA 的排序区是否为自动管理。

```
SQL> col name for a20
SQL> col value for a30
SQL> select name,value,isdefault
  2  from v$parameter
  3* where name in ('pga_aggregate_target','workarea_size_policy')

NAME                   VALUE                          ISDEFAULT
---------------------- ------------------------------ ---------------
pga_aggregate_target   200278016                      FALSE
workarea_size_policy   AUTO                           TRUE
```

从上例的输出可以看出参数 WORKAREA_SIZE_POLICY 是默认的参数，因为其 ISDEFAULT 的值为 TRUE，而参数 PGA_AGGREGATE_TARGET 是需要设置的，它不是系统的默认参数，因为 ISDEFAULT 的值为 FALSE。

下面我们分析一下当前数据库的 PGA 状态，如下例所示。

例子 6-73　通过数据字典视图 v$pgastat 查询 PGA 状态信息。

```
SQL> col name for a40
```

```
SQL> run
  1  select *
  2* from v$pgastat

NAME                                    VALUE       UNIT
------------------------------------ ----------- ------------
aggregate PGA target parameter          200278016   bytes
aggregate PGA auto target               16797888    bytes
global memory bound                     40054784    bytes
total PGA inuse                         19604480    bytes
total PGA allocated                     42294272    bytes
maximum PGA allocated                   90947584    bytes
total freeable PGA memory                      0    bytes
process count                                 21
max processes count                           33
PGA memory freed back to OS                    0    bytes
total PGA used for auto workareas              0    bytes

NAME                                    VALUE      UNIT
-------------------------------------- ---------- ------------
maximum PGA used for auto workareas     3831808    bytes
total PGA used for manual workareas           0    bytes
maximum PGA used for manual workareas         0    bytes
over allocation count                         0
bytes processed                       101315584    bytes
extra bytes read/written                      0    bytes
cache hit percentage                        100    percent
recompute count (total)                     869

已选择 19 行。
```

需要注意，以上输出中记录 PGA 状态的三个参数，即 aggregate PGA target parameter，aggregate PGA auto target 和 cache hit percentage，其含义说明如下。

- aggregate PGA target parameter：用户设置的当前 PGA 的内存总和。
- aggregate PGA auto target: Oracle 为 PGA 的排序区分配的内存大小，显然其值不能超过 PGA 内存的总和。
- cache hit percentage：说明排序区在 PGA 的排序区完成的比例，100 percent 表示全部排序都在 PGA 的排序区内完成，所以 Oracle 自动分配的排序区的尺寸是合理的。

我们可以使用以下方式监控 PGA 的排序区是否合理，其目的是观察参数 cache hit percentage 的值。如果是 100 percent 则认为 Oracle 根据系统状态设置的 PGA 的排序区是合理的，否则就需要增加参数 PGA_AGGREGATE_TARGET 的值，以满足为 PGA 的排序区添加内存的需要。监控 PGA 的排序区的方法如下例所示。

例子 6-74 查看在 PGA 的排序区进行排序的百分比。

```
SQL> select *
  2  from v$pgastat
```

```
  3  where name like 'cache%';

NAME                                      VALUE       UNIT
----------------------------------------  ----------  ---------
cache hit percentage                         100       percent
```

显然其中的 VALUE 值为 100，所以当前的排序行为都在 PGA 的排序区内完成。如果出现部分排序不在 PGA 的排序区完成，即 VALUE 的值小于 100，则需要考虑适当增加 PGA 的内存总和，如下例所示，通过设置参数 PGA_AGGREGATE_TARGET 来调整 PGA 的内存总和。

例子 6-75 调整 PGA 的内存大小 。

```
SQL> alter system set pga_aggregate_target = 76M;

系统已更改。
```

我们再查看修改结果，如下例所示。

例子 6-76 查看参数 PGA_AGGREGATE_TARGET 的修改结果。

```
SQL> show parameter pga

NAME                             TYPE          VALUE
------------------------------  ------------  ------------
pga_aggregate_target            big integer    76M
```

我们再次查看此时 PGA 的状态信息，看看 Oracle 为 PGA 的排序区分配了多少内存，如下例所示。

例子 6-77 查看 PGA 的大小以及 PGA 中排序区的大小。

```
SQL> select *
  2  from v$pgastat
  3  where name in ('aggregate PGA target parameter',
  4               'aggregate PGA auto target');

NAME                                      VALUE             UNIT
----------------------------------------  ----------------  -----------
aggregate PGA target parameter             78435456         bytes
aggregate PGA auto target                  22379217         bytes
```

可以看到此时的 PGA 的总和为 76M，同时自动分配给 PGA 的排序区的尺寸为 213.424805M，可见随着 PGA 的总内存的增加 Oracle 会自动增加其为排序区的分配的内存容量。

读者也可以通过数据字典视图 v$pga_target_advice 获得 PGA 的排序区进行排序行为的更详细的信息，如下例所示。

例子 6-78 查询 PGA 的排序区排序的详细信息。

```
SQL> select pga target for estimate/(1024*1024) as estd target,
  2  estd pga cache hit percentage ,estd overalloc count
  3  from v$pga target advice;

ESTD TARGET ESTD PGA CACHE HIT PERCENTAGE ESTD OVERALLOC COUNT
------------------- ------------------------------- -------------
```

```
          32                         100              0
          64                         100              0
          17                         100              0
         192                         100              0
          76                         100              0
  307.199219                         100              0
  358.399414                         100              0
  409.599609                         100              0
  460.799805                         100              0
         512                         100              0
         768                         100              0

ESTD TARGET ESTD PGA CACHE HIT PERCENTAGE ESTD OVERALLOC COUNT
----------------- ------------------------- --------------------
        1024                         100              0
        1536                         100              0
        2048                         100              0

已选择 14 行。
```

从以上输出看出，Oracle 给出了一系列的对 PGA 的估计值，并且给出了每个估计值状态下，排序在 PGA 的排序区完成的比率，由参数 ESTD_PGA_CACHE_HIT_PERCENTAGE 决定。由于笔者的计算机上没有任何排序行为，所以对于 PGA 的每一个估计值所有的排序都在 PGA 的内存中完成。如果在生产数据库系统上，参数 ESTD_PGA_CACHE_HIT_PERCENTAGE 一般不会都是 100，这样可以根据这个参数的提示，来确定手动设置的 PGA 内存总和是否合适。合适的含义是要求所有的排序行为都在 PGA 的内存中完成，这样通过内存排序减少了由于用于排序的内存不足而造成的 I/O 开销。

6.8 本章小结

Oracle 实例是一个非常重要的概念，它包含一组 Oracle 数据库相关的内存组件和一些后台进程。在实例优化中，我们先介绍了将程序或者数据常驻内存，这对于用户经常使用的程序如存储过程或函数、用户频繁访问的表等是十分有效的。

Oracle 实例的内存组件包括数据库高速缓存、共享池、重做日志缓冲区、PGA、大池和 Java 池。本章我们讨论了共享池的优化、重做日志缓冲区的优化、数据库高速缓存的优化以及 PGA 的优化，无论是那个组件的优化。我们首先需要理解这些组件的作用，然后学会如何调整这些组件的参数。在介绍如何优化时，我们也是按照这样的逻辑来说明，希望读者学习过程中，要时时回忆该组件的作用，这对于学习优化是十分重要的。

第 7 章

◂I/O以及系统优化▸

计算机的输入输出即 I/O 是很耗时的系统行为，I/O 优化就是通过一定的措施减少 I/O 消耗的时间。从本质上讲优化 I/O 的方法无非两种，一是最大化地减少 I/O 操作从而减少操作的数据量，二是平衡 I/O，如平衡数据库中各种文件的磁盘分布，它也同时减少了对统一磁盘文件的操作，减少了 I/O 量。本章讲述的 I/O 优化就是基于以上两点，只是针对不同粒度的数据库组件优化的具体方法不同。

7.1 I/O优化

在处理 I/O 优化问题之前，我们首先需要确认优化的对象是什么，那么如何确认这个对象呢？答案是使用数据字典视图。所以在面临 Oracle 的 I/O 相关的性能问题时，应该使用 v$system_event、v$session_event、v$session_wait 等视图查看系统事件信息、会话事件信息以及等待事件信息，从视图的数据中获得相关的性能瓶颈。这里再次强调读者在实施 Oracle 数据库优化时，应该对常用的等待事件比较熟悉，这样根据等待事件就可以初步判断导致等待的原因，采取迭代的方式实现优化。

读者应注意到，磁盘 I/O 的竞争是不可避免的，问题是这些磁盘 I/O 争用发生的频率，以及对于系统运行造成的影响。只有严重地影响数据库系统性能的 I/O 竞争才采取 I/O 优化措施。下面介绍不同的数据库组件的 I/O 优化原则和具体方法。

7.1.1 表空间 I/O 优化

在安装 Oracle 数据库时，所有表空间的数据文件都存放在相同的目录下，也都位于同一个物理磁盘上。显然这样的设置是不合理的，需要从 I/O 的角度进行优化，如将不同的表空间数据文件分布到不同的磁盘等。

Oracle 数据库在逻辑上划分成多个表空间，而表空间中包含物理的数据文件，这些文件是 Oracle 格式的操作系统文件，通过设计不同的表空间从而存放和管理不同的数据库文件。

下面分类介绍这些表空间、表空间作用以及相关的优化问题。表空间的分类如图 7-1 所示。

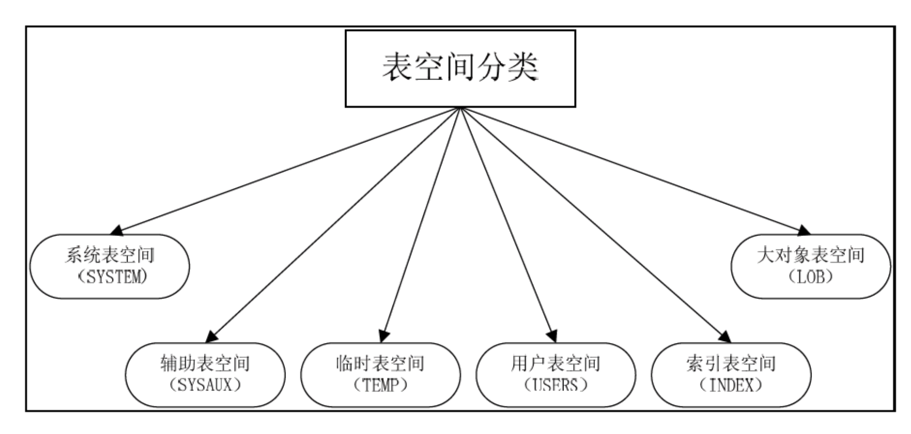


图 7-1　表空间的分类

1．系统表空间

系统表空间主要用于存放数据库字典信息，但是在默认情况下用户数据也存储在系统表空间中。如果将用户数据和数据字典存放在相同的表空间，显然存在磁盘 I/O 竞争，所以系统表空间中只存放和管理数据字典信息，对于用户数据可以创建用户表空间来单独存储。

2．辅助表空间

SYSAUX 表空间作为 SYSTEM 表空间的辅助表空间，以前使用独立表空间或系统表空间的组件，通过分离系统表空间的组件，减轻 SYSTEM 表空间的工作负荷，同时也避免了由于在 SYSTEM 表空间反复创建一些对象或者数据库组件造成的 SYSTEM 表空间的碎片。

3．还原表空间

还原表空间主要用于回滚用户更改的数据，如对于用户删除一个大表时（DROP），此时会产生大量的还原数据，对还原表空间造成一定的维护压力。而且对于 OLTP 系统，用户会频繁的更改表数据，还原表空间会不断地填充数据和释放磁盘空间，这样就比较容易产生磁盘碎片。所以还原表空间应该只用于还原数据，而不要用作其他用途。

4．临时表空间

临时表空间完成数据的磁盘排序行为，显然排序行为是数据库操作中比较频繁的一个数据库行为，这样就容易造成临时段的频繁分配和释放，也容易造成磁盘碎片，所以临时表空间不能存放永久数据，也不要存放如索引等其他数据库对象。

5．用户表空间

实际的数据库系统肯定需要存储用户数据，主要是表数据。表数据是应用程序或用户频繁操作的数据，需要单独建立一个表空间，这里关键是表空间中的数据文件要存储在单独的磁盘上。如果系统硬件资源不足，可以存储在 I/O 操作不频繁的磁盘上以减少 I/O 竞争产生的 I/O 等待。除了用户表空间，对于大型的表还需要建立索引，而索引和用户表最好放在不同的磁盘驱动器上，显然使用表的索引与搜索表数据同样存在竞争，所以最好创建索引表空间。即用户表空间我们分为用户数据表空间和用户索引表空间，将二者的数据文件存储在不同的磁盘上，可以有效地减少磁盘 I/O，从而优化 I/O 行为。

6．大对象表空间

大对象表空间是拥有大的数据库对象的表空间，该表空间的大小根据数据库块大小的不同，分布范围为 8TB~128TB。Oracle 数据库在处理数据时都需要将表数据从数据文件所在磁盘读取到数据库高速缓冲区中，这个过程是机械行为，所以相对于内存行为要消耗更多的时间，所以对于大对象表空间尤其需要考虑 I/O 优化。对于大对象表空间应该使用更大的块，使得每次读取数据时可以获得比小的块更多的数据，减少了磁盘 I/O。



上述对于表空间优化只给出了一个建议，总的思想是将不同表空间分布到不同的磁盘上，即将不同表空间的数据文件分开存放和管理，从而减少 I/O 竞争。具体的做法，如怎样将一个表空间的数据文件迁移到其他表空间，如何迁移系统表空间等操作可以参考本书前面关于表空间管理一章的内容。

7.1.2　数据文件 I/O 优化

我们知道在 Oracle 数据库中所有的数据都物理地存放在不同类型的数据库文件中，这些文件包括数据文件、数据文件、重做日志文件、控制文件，归档日志文件。当后台进程访问这些数据时，多个进程对同一个数据库文件的访问就造成 I/O 竞争，所以需要考虑对数据文件 I/O 优化问题，即考虑减少磁盘 I/O 或通过分布数据文件从而平衡 I/O 行为。下面我们分别介绍数据库文件 I/O 优化需要考虑的问题。

1．数据库文件 I/O 优化

- 数据文件优化：存放用户实际使用的表数据或索引等数据库对象，尤其对于大型的数据库系统必须创建单独的用户表空间，而不能使用默认的系统表空间。并且用户表空间的数据文件应分布在与系统表空间不同的磁盘上，在用户表空间中可以创建多个数据文件，并将这些数据文件存放在不同的磁盘上以平衡磁盘 I/O 行为。
- 控制文件优化：控制文件是 Oracle 数据库中非常重要的数据库文件，在数据库启动时告诉数据库它的数据文件、重做日志文件等文件的目录位置和用于数据库实例恢复的信息。出于系统可靠性考虑，应该将控制文件分别存放在不同的磁盘上，通过冗余的方式保证控制文件的可靠性。这样分布存放的同时也实现了控制文件的磁盘 I/O。
- 重做日志文件：在 Oracle 数据库系统中，要求必须设置两个重做日志组，而且每个日志组中至少有一个重做日志成员。但是在生产数据库中，应该创建不少于三个重做日志组，并且每个重做日志组不少于 2 个重做日志成员，这样可以保证重做日志文件的可靠性，并能提高数据库系统的效率（这当然还要涉及重做日志文件的大小设置问题）。Oracle 会循环使用重做日志组，在写一个重做日志组时，会同步写该组中的重做日志成员，所以同一个重做日志组中的重做日志成员应该存放在不同的磁盘上，以平衡数据 I/O。
- 重做日志文件和数据文件：重做日志文件和数据文件应该分开存放，即重做日志组中的每个成员和数据库系统中的数据文件应该分开存放，因为数据库写（DBWR）后台进程不但会操作数据文件，而且还会引发检查点使得重做日志写进程（LGWR）将数据库缓冲区中的脏数据同时写入重做日志文件。如果重做日志文件和数据文件放在同一个磁盘中，则存在 I/O 争用。

- 归档日志文件和重做日志文件：如果数据库处于归档模式，则重做日志文件在写满时切换到另一个重做日志组之前需要归档，即将当前重做日志文件中的数据全部写到归档文件中，这样保证了介质恢复的数据要求。显然如果二者放在同一个磁盘下，在读出重做日志文件的同时要写入归档文件，存在 I/O 的争用。在实际的数据库系统中，会造成重做日志切换时间过长，无法使用新的重做日志组而导致等待事件。所以应该将归档日志文件和重做日志文件分布到不同的磁盘上。

2. 数据库文件 I/O 监控

在实现了数据库文件的优化后，或许读者要问该如何监控数据文件的 I/O 状况呢？也就是说在数据库系统运行期间，如何通过某种方法知道当前数据库文件的信息，如输入输出量、读写消耗的时间等，答案是使用数据字典视图 v$filestat。

我们先查看该数据字典视图的结构，然后分析一些重要的列属性，如下例所示。

例子 7-1 查看数据字典 v$filestat 的结构。

```
SQL> desc v$filestat;
 名称                                  是否为空？ 类型
 --------------------------------- ------------- --------------------
 FILE#                                 NUMBER
 PHYRDS                                NUMBER
 PHYWRTS                               NUMBER
 PHYBLKRD                              NUMBER
 PHYBLKWRT                             NUMBER
 SINGLEBLKRDS                          NUMBER
 READTIM                               NUMBER
 WRITETIM                              NUMBER
 SINGLEBLKRDTIM                        NUMBER
 AVGIOTIM                              NUMBER
 LSTIOTIM                              NUMBER
 MINIOTIM                              NUMBER
 MAXIORTM                              NUMBER
 MAXIOWTM                              NUMBER
```

从输出看出，该数据字典的表结构中列数据类型都为数字类型 NUMBER，下面分析几个重要的列属性。

- FILE#：文件号说明对应的数据库文件标识，它和数据字典视图 dba_data_files 中的 file_id 含义相同，通过数据字典 dba_data_files 和 v$filestat 可以知道具体的一个数据文件的读取或写入统计数据。
- PHYRDS：说明物理读出该数据文件的数据块数。
- PHYWRTS：说明物理写入该数据文件的数据块数
- READTIM：说明读该数据文件使用的时间，单位是毫秒。
- WRITETIM：写入该数据文件使用的时间，单位是毫秒。

在使用数据字典 v$filestat 查看数据文件的操作统计数据前，需要设置系统参数 TIMED_STATISTICS 为 TRUE，这样数据字典 v$filestat 的 READTIM 和 WRITETIM 的值才不会是 0。查看该参数的值的方法，如下例所示。

例子 7-2　查看参数 TIMED_STATISTICS 的值。

```
SQL> show parameter timed_statistics;

NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ----------------
timed_statistics                      boolean             FALSE
```

从输出看出该值为 FALSE，所以必须使用 ALTER SYSTEM 指令将该参数设置为 TRUE，如下例所示。

例子 7-3　设置参数 TIMED_STATISTICS 的值为 TRUE。

```
SQL> alter system set timed_statistics = true;

系统已更改。
```

显然，输出提示参数已经更改，接下来查询修改结果，如下例所示。

例子 7-4　查看参数 TIMED_STATISTICS 的值。

```
SQL> show parameter timed_statistics;

NAME                                 TYPE        VALUE
------------------------------------ ----------- -------------
timed_statistics                      boolean     TRUE
```

现在可以使用数据字典 v$filestat 查看数据文件的 I/O 量信息了，如下例所示。

例子 7-5　使用数据字典 v$filestat 查看数据文件 I/O 量。

```
SQL>  select file#,phyrds,phywrts,readtim,writetim
  2  from v$filestat
  3  order by file#;

     FILE#     PHYRDS     PHYWRTS      READTIM    WRITETIM
------------ ---------- ------------ ---------- -----------
           1       5442          162      11572          52
           2         47           86       1511          90
           3        922          539       1339          39
           4         36            2         83           2
           5         10            2         73           2
```

从输出看出 FILE#为 1 的数据文件物理读数据块数最多，显然其读操作所消耗的时间也最长，排在第二位的是 FILE#为 3 的数据文件。下面我们再来查看数据字典 dba_data_files，它的 FILE_ID 和数据字典视图 v$filestat 中的 FILE#相同，如下例所示。

例子 7-6　查看数据库系统上的所有数据文件。

```
SQL> col file_id for 99
SQL> col file_name for a55
SQL> col tablespace name for a15
SQL> select file_id,file_name,tablespace_name
  2  from dba_data_files
```

```
  3* order by file_id

FILE_ID FILE_NAME                                        TABLESPACE_NAME
------- --------------------------- ------------ --------------------
1      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF   SYSTEM
2      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF  UNDOTBS1
3      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF    SYSAUX
4      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF      USERS
5      F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF    EXAMPLE
```

从结果看到共有 5 个数据文件，文件号从 1~5，FILE_ID 为 1 的文件的 SYSTEM01.DBF 文件，是系统数据库文件，位于系统表空间（SYSTEM）；而 FILE_ID 为 3 的文件的 SYSAUS.DBF 文件，是系统辅助数据库文件，位于辅助表空间(SYSAUX)。显然从文件 v$filestat 中得到的文件 I/O 量说明只有对数据库系统表空间的操作，而很少有对数据文件的操作，因为 FILE#为 4 的数据文件的读写操作很少，而且该文件是 USER01.DBF 文件，是为了保存用户数据而创建的表空间。

通过使用这两个数据字典视图，可以追踪数据文件粒度的 I/O 操作，知道哪个数据文件发生了 I/O 量大的操作。如果 I/O 是由多个 I/O 量大的数据文件引起，则需要平衡数据文件，将其分布到不同的磁盘上去，

7.1.3 表的 I/O 优化

对于表和索引的 I/O 优化，我们这里只给出一个优化思想，具体的操作读者可以参考创建表以及索引的相关章节。在商业数据库中，大表一般针对业务需求对一个或多个列建立索引，此时应该将索引存储的表空间和数据表存储的表空间分开，即两个表空间中的数据文件要放在不同的磁盘上，这样可以减少数据库使用索引查找数据时和数据文件的 I/O 竞争。

如果已经将表和该表的索引存储在同一个表空间的数据文件中，则可以通过迁移表，将该表移动到另一个表空间中。我们给出一个例子说明如何将一个表迁移到另一个表空间。先来查询 SCOTT 用户 EMP 表的存储表空间，如下例所示。

例子 7-7 查看 SCOTT 用户的 EMP 表的存储信息。

```
SQL> connect scott/tiger@orcl
已连接。
SQL> select table_name,tablespace_name
  2  from user_tables
  3  where table_name ='EMP';

TABLE_NAME                    TABLESPACE_NAME
------------------------- ------------------------------
EMP                             USERS
```

输出说明，当前表 EMP 的存储表空间为 USERS。为了不给读者增加负担，我们不再创建新的表空间，而是使用 SYSTEM 表空间作为新的表空间，将 EMP 表迁移到 SYSTEM 表空间，如下例所示。

例子 7-8　移动表 EMP 到新的表空间。

```
SQL> alter table emp move tablespace system;

表已更改。
```

上述输出表名，已经将表 EMP 移动到表空间 SYSTEM 中，下面我们使用数据字典 USER_TABLES 来验证移动结果，如下例所示。

例子 7-9　通过数据字典 USER_TABLES 来查看表 EMP 的表空间信息。

```
SQL> select table_name,tablespace_name
  2  from user_tables
  3  where table name ='EMP';

TABLE_NAME                         TABLESPACE_NAME
------------------------------ ------------------------------------
EMP                                 SYSTEM
```

显然，此时 EMP 表的存储表空间已经是 SYSTEM，说明我们成功移动表 EMP 到新的表空间中。

当然这里是为了演示方便，没有新建一个表空间，使用 SYSTEM 表空间代替，这显然是不合理的，所以下面我们将表 EMP 再次移动到 USERS 表空间。

例子 7-10　移动表 EMP 到 USERS 表空间。

```
SQL> alter table emp move tablespace users;

表已更改。
```

7.1.4　重建索引

通常建立索引的目的是加快表中数据的检索速度，但是建立索引对规模比较小的表效果是不明显的。而对于规模较大的表则不同，可以明显的提高表中数据的搜索速度，提高响应事件。

索引也是一个表记录或位图记录。Oracle 在使用索引时，同样需要对索引进行扫描需要的数据记录位置，再通过 ROWID 找到实际需要的表中的相关记录。但是在大型的 OLPT 数据库系统中，数据会频繁的插入或删除，如果删除的数据量很大，而用户又不断地搜索数据，此时就需要考虑对索引的 I/O 进行优化，原因是删除记录对应的索引并不会被 Oracle 删除而只是打上一个标记，这样造成索引记录中有大量的这样的标记，使得查询有效索引记录的时间增加。考虑一个极端的情况，一个 10000 行记录的索引，删除了其中 9900 行索引记录，为了使用剩余的 100 行有效地索引记录，最糟糕情况下需要扫描 10000 行索引记录，显然这降低了索引的使用效率，增加了索引操作的 I/O 数据量。此时为了避免这个问题，需要对索引进行重建。先查询一下 SCOTT 用户的 EMP 表上的索引，如下例所示。

例子 7-11　查看表 EMP 上的索引。

```
SQL> col owner for a10
SQL> col index name for a8
SQL> col table name for a10
SQL> set line 100
SQL> select owner,index name,index type,table name
  2  from dba_indexes
```

```
  3  where owner ='SCOTT'
  4* and table_name ='EMP'

OWNER      INDEX_NAME             INDEX_TYPE        TABLE_NAME
------------------ ------------- ----------------- ----------------------
SCOTT      PK_EMP                 NORMAL                 EMP
SCOTT      SCOTT_EMP_INCOME_IDX   FUNCTION-BASED NORMAL  EMP
```

可以看到在 SCOTT 用户的 EMP 表上有一个基于函数的索引和一个主键索引。我们假设基于函数的索引 SCOTT_EMP_INCOME_IDX 因为被删除了大量的索引记录而造成了使用索引的 I/O 性能下降。

首先需要进行索引的统计分析，如下例所示。

例子 7-12 使用 ANALYZE 指令分析索引。

```
SQL> analyze index scott.scott_emp_income_idx compute statistics;

索引已分析
```

然后，使用数据字典 INDEX_STATS 来查看删除索引相关的统计信息，如下例所示。

例子 7-13 使用数据字典 INDEX_STATS 来查看删除索引相关的统计信息。

```
SQL> select name,del_lf_rows,lf_rows_len
  2  from index_stats;

未选定行
```

因为索引 SCOTT_EMP_INCOME_IDX 中没有索引删除的记录，所以显示“未选定行”，数据字典 INDEX_STATS 中 NAME 的含义是索引的名字，DEL_IF_ROWS 的含义是删除的索引记录的长度，IF_ROWS_LEN 参数的含义是所有索引记录的长度。如果 DEL_IF_ROWS 占用 IF_ROWS_LEN 的 20%以上，则需要考虑重建索引，如下例所示。

例子 7-14 重建索引 SCOTT_EMP_INCOME_IDX。

```
SQL> alter index scott.scott_emp_income_idx rebuild;

索引已更改。
```

当使用 B 树索引时，如果索引的深度大于 3 也会影响使用索引的效率，此时也需要重建索引。我们先介绍如何查看索引的深度等信息。这里需要使用数据字典 DBA_INDEXES，如下例所示。

例子 7-15 查看索引的树深度信息。

```
SQL> select index_name,num_rows,tablespace_name,blevel,status
  2  from dba_indexes
  3* where table_name ='EMP'

INDEX_NAME             NUM_ROWS  TABLESPACE  BLEVEL  STATUS
------------------- ---------------- ----------- --------- ---------
PK_EMP                 12        USERS         0       VALID
SCOTT_EMP_INCOME_IDX   12        USERS         0       VALID
SCOTT_EMP_JOB_IDX       12       SYSTEM         0      UNUSABLE
```

输出显示结果中树的深度为 0，显然这样的索引是不需要优化的，如果通过查询认为有问题的索引，发现树的深度大于 3，则使用 ALTER INDEX 指令重建索引。参数 STATUS 说明当前索引的状态，VALID 说明该参数有效，如果该值为 INVALID，那也需要重建该索引。

7.1.5　迁移索引到新的表空间

如果在数据库系统运行初期，表和相应的索引存储在相同的表空间中，在系统运行后，依然可以通过索引迁移将这些索引迁移到不同的表空间。这些表空间的数据文件应该不在同一个磁盘驱动器上。当然，在迁移前读者已经知道需要迁移的索引，和要迁移到的表空间。下面我们将 USERS 表空间的 EMP 表的索引，迁移到新的表空间，具体步骤如下。

01 查询 EMP 表的索引以及所在表空间。使用 SCOTT 用户登录数据库，然后再查询索引的存储信息，如下例所示。

例子 7-16 查询表 EMP 的索引和对应的表空间。

```
SQL> col index_name for a10
SQL> col tablespace_name for a10
SQL>select index_name,table_name,status,tablespace_name
  2  from user_indexes
  3* where table_name='EMP'

INDEX_NAME TABLE_NAME                    STATUS   TABLESPACE
------------------ ------------------- --------------- ----------------
PK_EMP       EMP                           VALID     USERS
```

从输出可以看到表 EMP 的主键索引 PK_EMP 存储在 USERS 表空间，它和表 EMP 存储在相同的表空间。下面我们创建一个新的表空间，用于存储索引。

02 创建索引表空间，此时我们切换到 DBA 用户，并创建表空间，如下例所示。

例子 7-17 创建索引表空间。

```
SQL> conn sys/oracle as sysdba
已连接。
SQL> create tablespace index_tbs datafile 'd:/index.dbf'
  2  size 100m;

表空间已创建。
```

我们通过数据字典 user_tablespaces 查看是否成功创建所需要的表空间，如下例所示。

例子 7-18 验证表空间 index_tbs 的创建信息。

```
SQL> select tablespace name,status,contents
  2  from user tablespaces
  3  where tablespace name='INDEX TBS';

TABLESPACE STATUS    CONTENTS
---------- --------- ---------
INDEX_TBS  ONLINE    PERMANENT
```

输出显示表空间 INDEX_TBS 成功创建，状态为 ONLINE 并且是永久表空间。我们使用该表空间来存储索引。

我们在查看该表空间对应的数据文件的分布，必须保证该数据文件与表所在的表空间的数据文件分布在不同驱动器上，如下例所示。

例子 7-19 查询 INDEX_TBS 表空间的数据文件与其他数据文件的分布信息。

```
SQL> select ts#,name,status
  2  from v$datafile;

TS# NAME                                                        STATUS
---------- -------------------------------------------------- -------
 0   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF       SYSTEM
 1   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF      ONLINE
2   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF        ONLINE
4   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF         ONLINE
9   D:\INDEX.DBF                                                ONLINE
7   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\OLTBS.DBF           ONLINE
8   E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\OVRFLW.DBF          RECOVER

已选择 7 行。
```

从输出看出，表空间 INDEX_TBS 的数据文件为 D:/INDEX.DBF，显然该文件与其他数据文件不在同一个磁盘驱动器上。通过这些验证方法说明我们已经成功创建了所需要的索引表空间 INDEX_TBS。下一步我们迁移索引到该表空间。

03 迁移表 EMP 的索引 PK_EMP 到表空间 INDEX_TBS，如下例所示。

例子 7-20 迁移索引 PK_EMP 到新的表空间 INDEX_TBS。

```
SQL> alter index scott.pk_emp rebuild tablespace index_tbs;

索引已更改。
```

接下来我们验证是否成功迁移。先使用 SCOTT 用户登录，然后使用数据字典 USER_INDEXES 查询索引 PK_EMP 的存储表空间，如下例所示。

例子 7-21 查询表 EMP 的索引和对应的表空间。

```
SQL> conn scott/tiger
已连接。
SQL> select index_name,table_name,status,tablespace_name
  2  from user_indexes;

INDEX_NAME TABLE_NAME                     STATUS   TABLESPACE
---------- ------------------------------ -------- ----------
PK_DEPT    DEPT                           VALID    USERS
PK_EMP     EMP                            VALID    INDEX_TBS
```

输出说明表 EMP 的索引 PK_EMP 已经成功迁移到表空间 INDEX_TBS 中。这里也可以使用相同的方式将表迁移到一个新的表空间，如下例所示。

例子 7-22　迁移表到新的表空间。

```
SQL> alter table scott.emp move tablespace oltbs;

表已更改。
```

完成迁移后，通过数据字典 dba_tables 验证迁移结果，如下例所示。

例子 7-23　验证迁移结果。

```
SQL> select table_name,tablespace_name
  2  from dba_tables
  3  where owner='SCOTT';

TABLE_NAME                      TABLESPACE
------------------------------ ----------
DEPT                            USERS
EMP                             OLTBS
BONUS                           USERS
SALGRADE                        USERS
EMP_TEST                        USERS
NEW_TEST                        USERS

已选择 6 行。
```



在将表迁移到新的表空间后，要重建于该表相关的所有索引。因为此时的索引是 UNUSABLE 状态，如下例所示。

例子 7-24　查看索引状态。

```
SQL> select index_name,table_name,status
  2  from user_indexes;

INDEX_NAME TABLE_NAME                      STATUS
---------- ------------------------------ --------
PK_DEPT    DEPT                          VALID
PK_EMP     EMP                           UNUSABLE
```

从输出看出索引 PK_EMP 的状态是 UNUSABLE 状态，所以接下来需要重建该索引，如下例所示。

例子 7-25　迁移表后重建该表的索引。

```
SQL> alter index scott.pk_emp rebuild;

索引已更改。
```

向表中添加大量数据时可以先删除索引。

索引由 Oracle 数据库自己维护，当向表中添加数据行时，会同时更新索引，如果此时的数据量大，而且表的索引也比较复杂，此时可以考虑先删除该表上索引，在成功添加数据后再重建该索引。这在一定程度上可以减少由于索引更新而引起的 I/O 等待，减少数据的添加时间。

7.1.6 优化还原段

还原段的作用是用户发出 ROLLBACK 指令时还原用户对数据的更改，即如果用户执行了插入数据则执行删除操作，执行更新数据则修改回原始值，执行了删除操作则重新插入删除的记录。这些要恢复的数据放在还原段中，Oracle 为每个事务在还原表空间中分配一个还原段。

在删除一个大表时，还原数据可能会占满还原段的磁盘空间，从而导致数据库挂起，所以必须采取一定的措施。可使用 TRUNCATE 指令删除表中的数据，但保留表结构，此时不会产生还原数据，然后再使用 DROP 命令删除空表（表的结构）。下面举个例子说明，我们先创建一个模拟表，假设该表是大数据量的表，如下例所示。

例子 7-26 创建一个欲删除的模拟表。

```
SQL> create table large_table
  2  as
  3  select *
  4  from scott.emp;

表已创建。
```

验证该表是否成功创建，如下例所示。

例子 7-27 验证 LARGE_TABLE 表是否成功创建。

```
SQL> select table_name,status
  2  from user_tables
  3  where table_name like 'LAR%';

TABLE_NAME                      STATUS
------------------------------------------------------------ -----------------
LARGE_TABLE                     VALID
```

显然，表 LARGE_TABLE 创建成功，接着使用 TRUNCATE 删除该表中的数据，如下例所示。

例子 7-28 TRUNCATE 表 LARGE_TABLE。

```
SQL> truncate table large_table;

表被截断。
```

此时，表 LARGE_TABLE 的表结构还存在，继续使用 DROP 命令删除该表，这样就可以从系统中彻底删除大表 LARGE_TABLE 而不对还原段产生任何压力，如下例所示。

例子 7-29 DROP 表 LARGE_TABLE。

```
SQL> drop table large_table;

表已删除。
```

删除了表 LARGE_TABLE，此时该表的定义已经不存在了，如下例所示。

例子 7-30　查询表 LARGE_TABLE 的表结构。

```
SQL> desc large table;
ERROR:
ORA-04043: 对象 large_table 不存在
```

显然，已经不存在数据库对象 LARGE_TABLE，此时彻底删除了具有大数据量的表 LARGE_TABLE。

在商业数据库中有时需要删除大表中的一部分数据，而保留一部分认为有价值的数据。此时可以采用中介的方式实现，即将有用的数据拷贝入临时表，该表作为一个存储中介，然后 TRUNCATE 原表，再将临时表中的数据拷贝到原表，最后删除临时表，这样就不会产生大量的还原数据，减少还原段的 I/O 量。

我们可以使用 v$filestat 来查看 FILE#为 2 的文件的 I/O 量，如下例所示。

例子 7-31　查看数据文件 I/O 量。

```
SQL> select file#,phyrds,phywrts,readtim,writetim
  2  from v$filestat
  3  order by file#;

     FILE#     PHYRDS    PHYWRTS    READTIM   WRITETIM
---------- ---------- ---------- ---------- ----------
         1      5696        85      11747        57
         2        47       443       1511        90
         3       967       793       1382        58
         4        38         2         84        2
         5        10         2         73        2
```

上例的输出结果与例子 7-5 相比， FILE#为 2 数据文件的 I/O 量几乎没有什么变化，这个功劳应该归功于 TRUNCATE 的作用，因为它是 DDL 语句，不产生还原段数据。

7.2　优化操作系统

Oracle 作为一个数据库系统是运行在操作系统上的，无论如何它不能脱离操作系统的环境，所以收集某个时间段中核心数据是有帮助的，如收集 CPU 的使用、检测系统 I/O 以及监控内存使用等。下面我们从两种操作系统平台开始介绍如何监控系统资源，以确定优化 Oracle 的当前 OS 瓶颈。

7.2.1　在 WINDOWS 平台启动监控

在 WINDOWS 平台上使用“任务管理器”来监控系统中运行的进程状态，以及系统的性能问题。性能包括 CPU 的使用、内存的使用率，以及 CPU 的使用频率图。因为 WINDOWS 良好的窗口特性，读者应该很熟悉“任务管理器”，同时按住 Ctrl+Alt+Delete 组合键启动“任务管理器”，单击“进程”标签如图 7-2 所示，显示当前运行的进程、占用的 CPU 和内存资源。单击“性能”标签如图 7-3 所示，显示 CPU 以及内存使用的动态图。



图 7-2　使用资源管理器监控进程

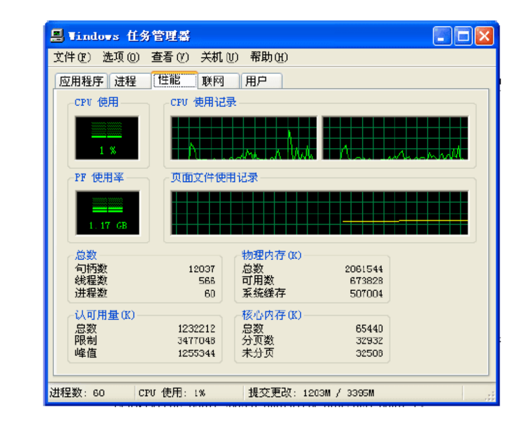


图 7-3　使用资源管理器监控性能

在使用资源管理器监控进程信息时，很容易确定哪个进程占用了最多的 CPU 以及占用了最多的内存等信息，从而定位哪些进程占用了这些宝贵的资源。一般在运行了 Oracle 的数据库服务器上，Oracle 是占用系统资源（CPU 和内存）最多的进程，Oracle 在 WINDOWS 平台上显示为一个进程，而我们熟悉的实例的后台进程在 WINDOWS 平台上以线程的形式出现，这与 UNIX 系统不同。

由于 WINDOWS 中监控 OS 性能的工具容易使用，直观简洁，所以这里不做过多介绍。下面详细介绍在 UNIX 系统上实现确定 OS 瓶颈的性能监控。

7.2.2　UNIX 系统上实现性能监控

本节将介绍在 UNIX 系统上如何监控 CPU 的使用情况、设备使用情况、以及虚拟内存的使用情况，最后简单介绍有关网络连接数据的统计信息。

7.2.3　监控 CPU 的使用情况

在 Sun 的 Solaris 操作系统上可以使用 SAR 指令监控 CPU 的使用情况，SAR 的意思是“系统活动报告”（system activity reporter）。可以使用-u 参数查看 CPU 的使用信息，下面先给出一个例子，然后解释其作用和相关参数的含义，如下例所示。

例子 7-32　使用 sar –u 指令监控 CPU 的使用情况。

```
root@BJbackup # sar -u 5 10

SunOS BJbackup 5.6 Generic_105181-34 sun4u    09/8/09

16:23:23    %usr    %sys    %wio   %idle
16:23:33      8      10       2      63
16:23:38     15       9       2      74
16:23:43     14       8       1      77
16:23:48     14       9       1      75
16:23:53     14      10       1      75
16:23:58     11      10       0      78
```

```
16:24:03      16      10       1      73
16:24:08      16       9       2      73
16:24:13      14       9       0      77

Average       15       9       1      74
```

上述通过执行 sar –u 5 10 命令监控测量了 CPU 的使用，其中-u 表示测量 CPU，第一个参数 5 表示测量周期，第二个参数 10 表示测量的次数，指令 sar –u 5 10 的含义就是每隔 5 秒钟测量一次 CPU 的使用情况，检测 10 次。

下面解释 CPU 的使用情况的意义。第一个参数%usr 表示用户进程使用的 CPU 的百分比，在 Solaris 系统上 Oracle 服务器进程被认为是用户进程。第二个参数%sys 表示操作系统自身消耗的 CPU 的百分比，如 CPU 的上下文环境切换、中断服务等。第三个参数%wio 表示等待 I/O 的进程占用的 CPU 的百分比，显然这个数值越高不是好事，说明系统效率很低，大量的进程占用了 CPU 资源等待 I/O，这些 CPU 时间片都用于进程的上下文切换了。第四个参数%idle 表示空闲 CPU 的百分比，说明这些 CPU 资源可用，从资源利用率的角度看，此值越小越好，这样 CPU 资源得以充分利用。

注意，如果 CPU 的%idle 值为 0%并不意味着瓶颈出现。这要看等待使用 CPU 的队列长度，一般如果这个长度小于 CPU 个数的 2 倍，此时 CPU 的空闲率为 0%，不表示系统存在 CPU 的资源瓶颈。下面使用 sar –q 指令查看执行队列的长度，以及正在使用的 CPU 的百分比。

例子 7-33 使用 sar –q 查看 CPU 的使用情况。

```
root@BJbackup # sar -q 5 10

SunOS BJbackup 5.6 Generic_105181-34 sun4u    09/8/09

16:8:07 runq-sz %runocc swpq-sz %swpocc
16:8:12
16:8:17     1.0      20
16:8:22
16:8:27
16:8:32     1.0      20
16:8:37
16:8:42
16:8:47
16:8:52
16:8:57

Average      1.0       4
```

指令 sar –q 5 10 的含义是每隔 5 秒钟，查看 CPU 的执行队列等信息，测量 10 次。从上例的输出结果可以看出，在 10 次测量中不是每次各个参数都有数值。我们先分析参数的含义，第一个参数 runq-sz 说明当前 CPU 的执行队列中进程的数量，第二个参数%runocc 说明正在运行的 CPU 占用的百分比，而其他两个参数 swpq-sz 和参数%swpocc 表示交换队列的信息。

显然上述输出显示在测试时间段内，系统中 CPU 的执行队列中的进程数都为 1，多数情况下，根本没有排队。结合上例中的 CPU 的%idle 数值说明当前系统的 CPU 资源不存在瓶颈，而且可以说 CPU 资源很充分。而正在运行的 CPU 平均值只占 4%。也说明当前的 CPU 比较空闲。

7.2.4 监控设备使用情况

遇到与系统 I/O 有关的瓶颈问题，需要分析设备的使用情况，此时可以使用 sar –d 指令来测量设备的使用信息，如下例所示。

例子 7-34 使用 sar –d 测量和设备相关的信息。

```
Croot@BJbackup # sar -d 5 3

SunOS BJbackup 5.6 Generic_105181-34 sun4u    09/8/09

16:20:30   device       %busy   avque   r+w/s  blks/s  avwait  avserv

16:20:35     nfs1        0        0.0      0       0     0.0      0.0
             sd0         0        0.0      0       0     0.0      0.0
             sd0,a       0        0.0      0       0     0.0      0.0
             sd0,b       0        0.0      0       0     0.0      0.0
             sd0,c       0        0.0      0       0     0.0      0.0
             sd0,g       0        0.0      0       0     0.0      0.0
             sd0,h       0        0.0      0       0     0.0      0.0
             sd1         3        0.0      7     112      0.0      4.9
             sd1,a       3        0.0      7     112      0.0      4.9
             sd1,c       0        0.0      0       0     0.0      0.0
             sd6         0        0.0      0       0     0.0      0.0
...............为了节省篇幅，省略了两次测量结果。
Average    nfs1          0        0.0      0       0     0.0      0.0
         sd0             0        0.0      0       1     0.0     10.6
         sd0,a           0        0.0      0       1     0.0     10.6
         sd0,b           0        0.0      0       0     0.0      0.0
         sd0,c           0        0.0      0       0     0.0      0.0
         sd0,g           0        0.0      0       0     0.0      0.0
         sd0,h           0        0.0      0       0     0.0      0.0
         sd1             3        0.0      6      89     0.0      4.9
         sd1,a           3        0.0      6      89     0.0      4.9
         sd1,c           0        0.0      0       0     0.0      0.0
```

指令 sar –d 5 3 表示测量设备的使用情况三次，每次间隔为 5 秒，我们解释一下测量结果中的参数含义，第一个 device 为设备名，第二个%busy 为给定设备的繁忙百分比，第三个参数 avque 设备的平均队列长度，第四个参数 r+w/s 每秒该设备的读出和写入的数据，第五个参数 blks/s 该设备每秒传输的数据量，第六个参数 avwait 表示设备 I/O 操作所用的平均时间，第七个参数 avserv 表示 5 秒周期内每个 I/O 操作的平均等待时间。

在上述输出中应该注意%busy 的值。如果该值超过 60%，一般表示该设备有问题，需要检测确认磁盘状况，当然平均队列长度，平均等待时间都是和系统的%busy 相关的特性。还有就是 I/O 操作的平均使用时间不应该超过 100 毫秒，否则需要检测磁盘以找出原因。

7.2.5 监控虚拟内存使用情况

我们使用 vmstat 指令获得虚拟内存的统计信息，如 vmstat 5 10，表示每 5 秒钟测试一次虚拟内存的使用，测量 10 次，指令执行结果如下例所示。

例子 7-35 查看虚拟内存的使用信息。

```
root@BJbackup # vmstat 5 10
 procs     memory            page     disk           faults         cpu
 r b w swap  free   re  mf pi  po fr de  sr s0 s1  s6 --    in   sy   cs  us sy id
 0 0 0   928  820    0  0  0  0  0  0  0  0  5  0  0 484967196 0 0 -14 -9 -103
 0 0 0 3446016 556288  0 1855 0  0  0  0  0  0  4  0  0  612 3159  578 16  9 76
 0 0 0 3445864 556496  0 1884 0  0  0  0  0  0  5  0  0  599 3102  549 12 10 78
 0 0 0 3446208 556432  0 1842 0  0  0  0  0  0  7  0  0  634 3272  602 15 10 76
 0 0 0 3446344 556472  0 1873 0  0  0  0  0  0  4  0  0  600 3149  561 14 10 77
 0 0 0 3445616 556392  0 1844 0  0  0  0  0  0  4  0  0  595 3027  536 12  9 79
 0 0 0 3445472 556208  0 1839 0  0  0  0  0  3 10  0  0  677 3377  590 20 10 70
 0 0 0 3446392 556568  0 1849 0  0  0  0  0  0  6  0  0  617 3206  574 14 10 77
 0 0 0 3446136 556320  0 1846 0  0  0  0  0  0  9  0  0  68  3135  555 13  9 79
 0 0 0 3446496 556576  0 1380 0  0  0  0  0  0  5  0  0  569 2452  464 10  8 83
 0 0 0 3446336 556472  0 1460 0  0  0  0  0  0 10  0  0  599 872  477 12  8 80
```

虚拟内存状态信息的输出中包括六个部分，说明如下：

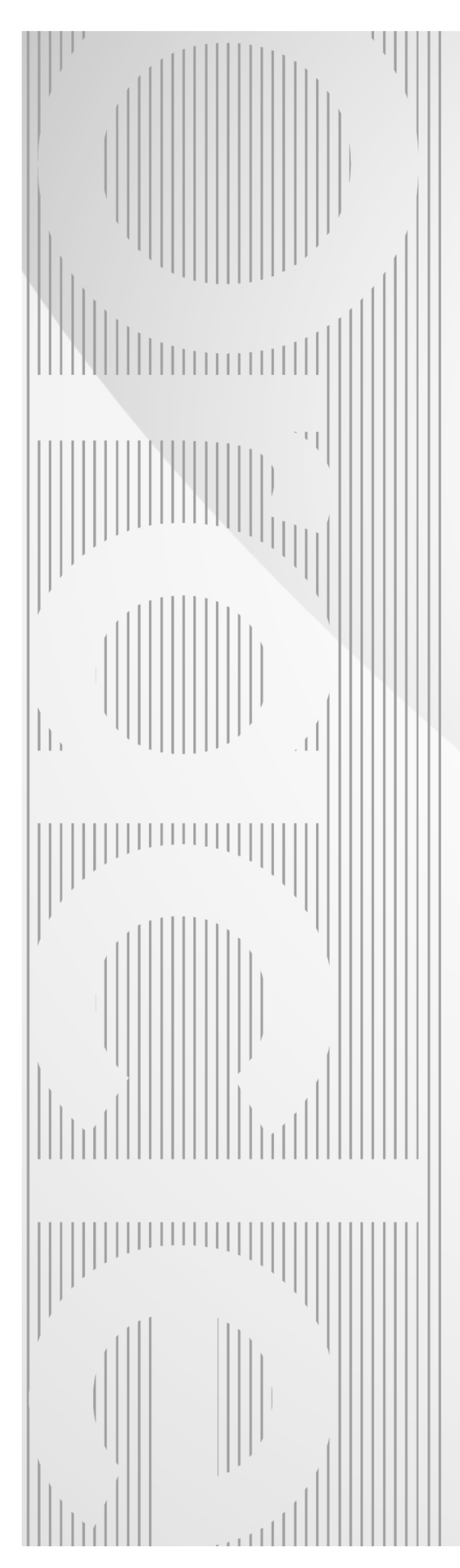
- PROCS：说明进程信息。其中 r 和 b 代表执行队列和阻塞队列的大小，r 值应该小于 2 倍的 cpu 大小，否则存在 CPU 瓶颈，CPU 无法正常处理排队的等待进程，b 说明阻塞队列的长度，此时的阻塞一般是由 I/O 引起，阻塞对象的长度可以及时的反映系统的性能；
- MEMORY：说明这些进程的内存使用状态。SWAP 说明当前可用的交换空间，单位是 K 字节，FREE 说明内存中自由表的大小，同样以 K 字节为单位。
- PAGE：说明进程换页信息。pi 和 po 说明调入内存页和调出内存页的字节数，单位为 K 字节。Fr 说明空间的 K 字节数，de 说明预期的短期内存不足的字节数，sr 说明由时钟扫描算法扫描的页面数，页面大小由操作系统确定，总之 si,so 提供页交换信息，一般如果没有过渡分配 SGA 或 PGA 给数据库，则该值一般为 0，de 和 sr 以 K 字节为单位说明系统是否缺乏内存。
- DISK：说明磁盘使用信息。最好使用 sar –d 来查看更多的磁盘使用情况。
- FAULT：系统范围内的陷阱或故障信息。其中 in 表示设备中断数，sy 表示系统调用数，cs 表示 CPU 环境切换的数目。
- CPU：说明了 CPU 的使用信息。其中 us 说明用户进程使用 CPU 的百分比，sy 说明系统进程使用时间的百分比，id 说明非当前系统或用户进程使用的时间的百分比，如所有的 I/O 等待占有的 CPU 时间。

在 UNIX 系统的优化中，我们主要是使用相关的指令，查看系统各种资源的使用情况，找出影响数据库运行效率的 OS 组件，如 CPU 瓶颈、系统内存不足等来优化数据库行为。

7.3 本章小结

在任何计算机上，磁盘读非常占用系统时间，往往造成 CPU 等待或者系统应用的等待。随意平衡 I/O 是数据库优化的重要方面，在 Oracle 数据库中主要涉及表空间优化、数据文件优化，表优化、索引优化以及还原端优化。这些数据库对象的优化思想都是尽量减少磁盘读取的次数和平衡 I/O 数据量，减少磁盘读取造成的等待。

Oracle 数据库从宏观来讲就是一个应用软件，它安装在操作系统上，所以操作系统自身的性能无疑会影响数据库的性能。因此，监视操作系统的性能以及解决操作系统的性能问题，也是 DBA 在优化数据库时需要认真对待的。



·第3部分·

数据库备份与恢复

本部分包括第 8~12 章，详细介绍了 Oracle 的所有备份和恢复工具以及具体实例。其中对 RMAN 引入数据块恢复的案例，并介绍了传输表空间的概念以及使用 EXP、IMP 传输表空间的实作案例。本部分也详细介绍了 Flashback 数据库技术。

第 8 章

RMAN备份与恢复数据库

RMAN 是 Oracle 提供的程序 Recovery Manager，即恢复管理器。使用 RMAN 可以轻松实现数据库的所有备份任务，如备份整个数据库、特定的表空间或者数据文件。RMAN 是 Oracle 提供的一个更加智能和自动化的备份恢复工具，具有许多新的特性如实现增量备份、备份文件的差错检验等。本章将详细介绍 RMAN 技术的每一个细节。

8.1 RMAN概述

RMAN 在数据库服务器的帮助下实现数据库文件、控制文件、数据库文件和控制文件的映像副本、归档日志文件，以及数据库服务器参数文件的备份。RMAN 也允许使用脚本文件实现数据的备份与恢复，而且这些脚本保存在数据库内，不需要编写基于 OS 的脚本文件。RMAN 备份的文件自动保存在一个系统指定的目录下，文件的名称也由 RMAN 自己维护。当实现数据恢复操作时，恢复指令简洁，RMAN 自动寻找需要的文件实现数据恢复，减少了在传统的导出导出程序中人为错误的发生。本节将详细介绍 RMAN 备份与恢复的技术细节，包括 RMAN 的优点、RMAN 的系统结构、快闪恢复区、使用 RMAN 脚本、RMAN 参数配置、恢复目录的概念，以及使用 RMAN 实现数据备份和数据恢复。

8.2 RMAN的独特之处

如果读者使用过 EXP/IMP 以及 EXPDP/IMPDP 工具，应该能很好地理解使用 RMAN 带来的好处，Oracle 每次技术的演进都是使其功能更强大，操作更简单，更加满足生产数据库的要求。相对“古老”的备份技术，使用 RMAN 具有如下的优点。

- 支持增量备份：在传统的备份工具中如 EXP 或 EXPDP，只能实现一个完整备份而不支持增量备份。RMAN 采用被备份级别实现增量备份，在一个完整备份的基础上，采用增量备份。与传统的备份方式相比，这样可以减少备份的数据量。
- 自动管理备份文件：RMAN 备份的数据是 RMAN 自动管理的，包括文件名字、备份文件存储目录。它能识别最近的备份文件，搜索恢复时需要的表空间、模式或数据文件等备份文件。

- 自动化备份与恢复：在备份和恢复操作时，使用简单的指令就可以实现备份与恢复，且执行过程完全由 RMAN 自己维护。
- 不产生重做信息：与用户管理的联机备份不同，使用 RMAN 的联机备份不产生重做信息。
- 恢复目录：RMAN 的自动化备份与恢复功能应该归功于恢复目录的使用，RMAN 直接在其中保存了备份和恢复脚本。
- 支持映像拷贝：使用 RMAN 也可以实现映像拷贝，映像是以操作系统上的文件格式存在，这种拷贝方式类似于用户管理的脱机备份方式。
- 新块的比较特性：这是 RMAN 支持增量备份的基础，这种特性使得在备份时，跳过数据文件中从未使用过的数据块的备份。备份数据量的减少直接导致了备份存储空间要求和备份时间的减少。
- 备份的数据文件压缩处理：RMAN 提供一个参数，说明是否对备份文件进行压缩，压缩的备份文件以二进制文件格式存在，可以减少备份文件的存储空间。
- 备份文件有效性检查功能：这种功能验证备份的文件是否可用，在恢复前往往需要验证备份文件的有效性。

8.3　RMAN系统架构详解

Oracle 的 RMAN 工具使用会话建立客户端到数据库服务器的连接。用户首先需要启动 RMAN 可执行程序，然后建立客户端与服务器端的会话连接，用户通过 RMAN 的客户端进行 RMAN 操作，执行备份与恢复指令，这些指令在服务器端的服务器进程中执行，而服务器进程完成实际的磁盘读写操作。下面我们详细介绍 RMAN 的系统架构组成。

- RMAN 可执行程序：它是一个客户端工具，用来启动与数据库服务器的连接，从而实现备份与恢复的各种操作。
- RMAN 客户端：一旦建立了与数据库服务器的会话连接，RMAN 可执行程序就创建一个客户端，通过客户端完成与数据库服务器之间的通信，完成各种备份与恢复操作的指令。RMAN 客户端可以通过 Oracle Net 连接到可访问的任何主机上。
- 服务器进程：在 RMAN 建立了与数据库服务器的会话连接后，在数据库服务器端启动一个后台进程，它执行 RMAN 客户端发出的各种数据恢复与备份指令，并完成实际的磁盘或磁带设备的读写任务。
- RMAN 信息库：RMAN 信息库记录了 RMAN 的一些信息，如备份的数据文件及副本的目录、归档的重做日志备份文件和副本、表空间和数据文件，以及备份或恢复的脚本和 RMAN 的配置信息。默认使用数据库服务器的控制文件记录这些信息，读者可以通过转储的控制文件发现这些信息，如使用 ALTER DATABASE BACKUP CONTROL FILE TO TRACE。
- 恢复目录：记录 RMAN 信息库的信息。但是恢复目录需要事先配置，信息库既可以存储在数据库的控制文件中，也可以存储在恢复目录中。在 Oracle 中默认先将 RMAN 信息库写入控制文件，如果存在恢复目录则需要继续写到恢复目录。使用控制文件的不足是控制文件中记录 RMAN 信息库的空间有限，当空间不足时可能被覆盖掉。所以 Oracle 建议创建单独的恢复目录，这样也可以更好的发挥 RMAN 提供的新特性。

注意

图 8-1 给出了 RMAN 的系统结构图，其实也可以理解为一个备份或恢复过程的信息流示意图，RMAN 可执行程序启动并建立与数据库服务器的会话连接，客户端发出备份指令，而数据库服务器端的服务器后台进程执行指令完成磁盘读写操作，并将备份信息记录在 RMAN 信息库中，RMAN 信息库可以保存在数据库服务器端的控制文件中。如果使用恢复目录，RMAN 信息库同样会自动保存在恢复目录中。

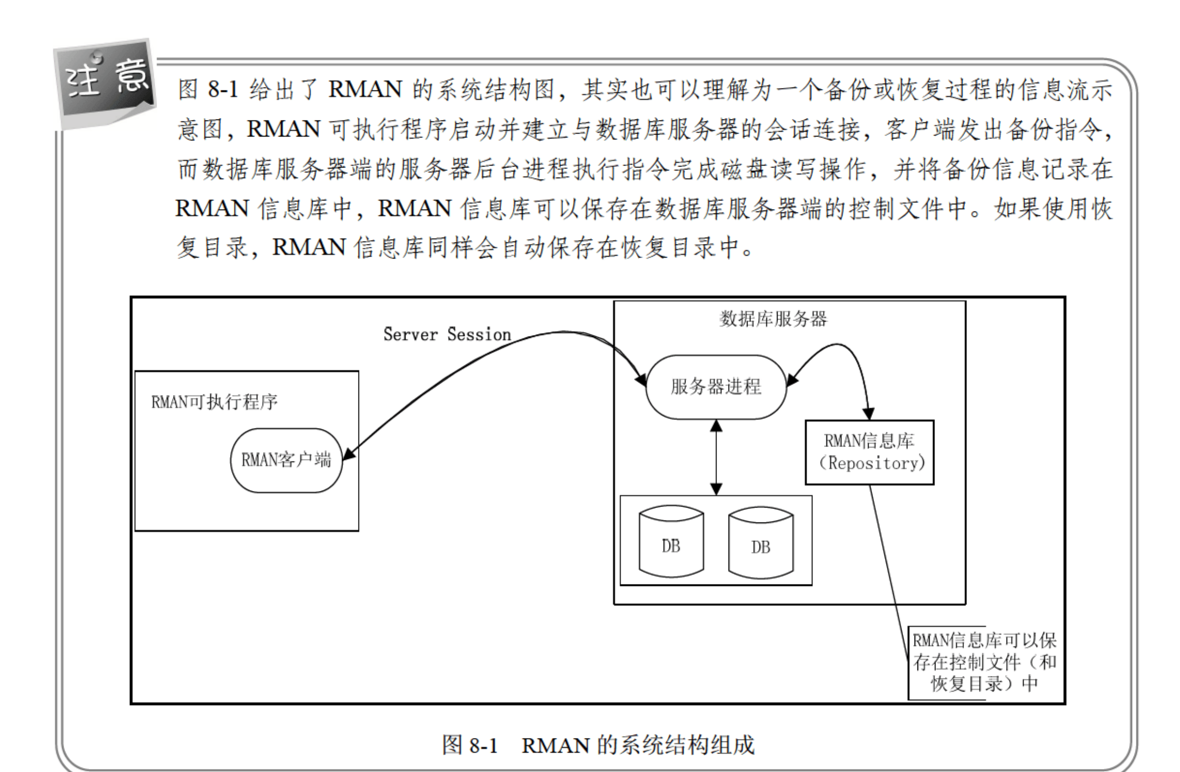


图 8-1　RMAN 的系统结构组成

8.4　快闪恢复区（flash recovery area）

快闪恢复区是存储备份恢复数据文件以及相关信息的存储区。快闪恢复区保存了每个数据文件的备份、增量备份、控制文件备份以及归档重做日志备份。Oracle 也允许在快闪恢复区中保存联机重做日志的冗余副本，以及当前控制文件的冗余副本，还有 Oracle 闪回特性中的闪回日志也保存在快闪恢复区中。

在使用 RMAN 实现数据库的备份与恢复时，配置的快闪恢复区就是 RMAN 存储所有与备份相关的文件存储区，而此时的文件名不需要用户干预，Oracle 使用 OMF 创建备份文件的文件名。

使用快闪恢复区的优点是，实现了备份文件的自动管理，使得备份与恢复数据库更简单（指令更简洁），并且可以集中管理磁盘空间。

8.4.1　修改快闪恢复区大小

那么如何管理快闪恢复区呢？在 Oracle 中快闪恢复区由两个初始化参数设置，一个是 DB_RECOVERY_FILE_DEST_SIZE，该参数用于设置快闪恢复区的最大容量，另一个是 DB_RECOVERY_FILE_DEST，该参数设置快闪恢复区在操作系统磁盘空间上的位置。可以通过两种方式来设置快闪恢复区的参数，一种方法是在初始化参数文件 init.ora 文件中设置这两个参数，另一种方法是通过数据库指令 ALTER SYSTEM 在运行的数据库上动态地设置它们。下面演示如何

动态设置快闪恢复区的参数。

首先查看当前数据库的快闪恢复区参数，使用 SHOW PARAMETER 指令，如下例所示。

例子 8-1　查看快闪恢复区的参数信息。

```
SQL> show parameter db recovery

NAME                         TYPE          VALUE
---------------------------- ------------- -------------------------
db_recovery_file_dest        string        F:\oracle\product\10.2.0/flash
                                            recovery area
db_recovery_file_dest_size   big integer   2G
```

从中我们可以看到快闪恢复区在磁盘上的目录和快闪恢复区的空间大小，我们备份的整个数据库以及控制文件都保存在该快闪恢复区中。该区域中的文件由 Oracle 自己维护，一旦需要恢复数据库时，只需要使用简单地指令就可以恢复数据库。RMAN 工具会自动寻找存储在快闪恢复区中的备份文件完成恢复。



快闪恢复区的参数可以动态更改，如可以在数据库运行期间改变快闪恢复区的大小，以及改变快闪恢复区在磁盘上的存储目录。

下面演示如何动态修改快闪回复区的大小，如下例所示。

例子 8-2　修改快闪恢复区的参数。

```
SQL> alter system set
  2  db recovery file dest size=2g;

系统已更改。

SQL> alter system set
  2  db recovery file dest = 'f:\flashrecovery area'

系统已更改。
```

快闪恢复区的参数除了在运行库上动态更改，或者在 init.ora 文件或 SPFILE 文件中设置，也可以使用 OEM 工具的 DATABASE CONTROL 配置快闪恢复区。

为了以后演示方便，这里仍将快闪恢复区的目录设置为其默认目录，大小仍为 2G。

如果不需要快闪恢复区，可以将参数 DB_RECOVERY_FILE_DEST 的值设置为空格，使得快闪恢复区不存在存储目录。

当使用了快闪恢复区后，可以通过数据字典 v$recovery_file_dest 来查看快闪恢复区的空间使用情况以及文件数量，如下例所示。

例子 8-3　查看快闪恢复区的位置以及空间使用信息。

```
SQL> col name for a30
SQL> set line 100
SQL> select name,space limit,space used,number of files
  2* from v$recovery_file_dest
```

```
NAME                          SPACE_LIMIT SPACE_USED NUMBER_OF_FILES
------------------- --------------- --------------- ------------------
F:\oracle\product\10.2.0/flash     2147483648   793569280                7
_recovery_area
```

上述输出说明当前数据库的快闪恢复区的空间限制为 2G，已经使用了 756M，当前恢复区中的文件数为 7。NAME 的值说明快闪恢复区的操作系统目录，该目录为 F:\oracle\product\10.2.0\flash_recovery_area。

8.4.2　解决快闪恢复区的空间不足问题

如果快闪恢复区的空间不足该如何处理呢，有三种方法可以处理：一是增加恢复区磁盘空间，但这受当前磁盘空间的限制，二是删除没用的备份文件或将备份文件拷贝到磁带设备，三是删除当前的恢复区，重新设置新的快闪恢复区。

- 增加磁盘空间，可以使用 ALTER SYSTEM 指令动态设置快闪恢复区的空间大小，如下例所示。

例子 8-4　重新设置快闪恢复区的空间大小。

```
SQL> alter system set
  2  db_recovery_file_dest_size=4g;

系统已更改。
```

- 使用 CROSSCHECK 和 DELETE EXPIRED 指令删除不需要的文件。使用 RMAN 的 BACKUP RECOVERY AREA 指令将恢复区中的文件拷贝到磁带中。
- 删除当前的快闪恢复区，重新设置。

```
SQL> alter system set db_recovery_file_dest ='f:\newflasharea'
```



当向快闪恢复区添加新文件时，Oracle 会自动更新文件列表，发现符合删除条件的备份文件，这些文件包括不符合保留策略的文件，及拷贝到磁带的过渡文件。而重做日志文件和控制文件任何时候都不会被删除。

8.5　建立RMAN到数据库的连接

前面已经介绍了 RMAN 的基本功能、系统组成，以及使用 RMAN 实现备份与恢复的快闪恢复区，本节讲述如何使用 RMAN 建立到数据库服务器的连接。我们通过例子说明连接到数据库服务器，如下例所示。

例子 8-5　使用数据库用户名和密码登录 RMAN。

```
D:\>rman

恢复管理器: Release 11.1.0.6.0 - Production on 星期一 8月 31 22:10:05 2009
```

```
Copyright (c) 1982, 2007, Oracle.  All rights reserved.

RMAN> connect target system/oracle@orcl

连接到目标数据库: ORCL (DBID=121982901)
```

上例说明首先在操作系统环境下输入 RMAN 指令，启动 RMAN 可执行程序，然后使用 connect target 指令使用数据库用户名和密码建立与数据库服务器的会话连接，当然，也可以使用 rman target 指令来建立这个会话连接，如下例所示。

例子 8-6　使用操作系统认证连接到 RMAN。

```
D:\>rman target /

恢复管理器: Release 11.1.0.6.0 - Production on 星期一 8 月 31 22:12:41 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

已连接到目标数据库: LEEJIA (DBID=587536714)
```

8.6　RMAN实现脱机备份

下面通过实例说明如何实现 RMAN 的脱机备份，要实现脱机备份首先需要使用 RMAN 登录到数据库服务器，关闭数据库，然后启动数据库到 MOUNT 状态，再执行 BACKUP DATABASE 指令备份整个数据库，具体步骤说明如下。

01 使用数据库用户名和密码登录 RMAN。

```
D:\>rman target system/oracle@orcl
恢复管理器: Release 11.1.0.6.0 - Production on 星期六 8 月 29 16:48:49 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
连接到目标数据库: ORCL (DBID=121982901)
RMAN>
```

02 在 RMAN 执行程序中，通过客户端指令关闭数据库，然后从 RMAN 加载数据到 MOUNT 状态。

```
RMAN> shutdown immediate
使用目标数据库控制文件替代恢复目录
数据库已关闭
数据库已卸载
Oracle 实例已关闭
RMAN> startup mount
已连接到目标数据库 (未启动)
Oracle 实例已启动
数据库已装载
系统全局区域总计     603979776 字节

Fixed Size                  190380 字节
Variable Size             29495412 字节
```

```
Database Buffers            369098752 字节
Redo Buffers                  7135232 字节
```

03 使用 BAKCUP DATABASE 备份指令备份整个数据库，如没有配置快闪恢复区，则需要使用 FORMAT 参数说明要备份的全库的备份集放在哪个目录下，如下例所示。

```
RMAN> backup database;

启动 backup 于 01-9 月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00002
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
输入数据文件 fno=00001
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
输入数据文件 fno=00005
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
通道 ORA_DISK_1: 正在启动段 1 于 01-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_
01\O1 MF NNNDF TAG20090901T223404 59TD6X2G .B
KP 标记=TAG20090901T223404 注释=NONE
通道 ORA DISK 1: 备份集已完成, 经过时间:00:02:46
完成 backup 于 01-9 月 -09

启动 Control File and SPFILE Autobackup 于 01-9 月 -09
段
handle=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\AUTOBACKUP\2
009 09 01\O1 MF S 696465171 59TDD3LK .BKP commen
t=NONE
完成 Control File and SPFILE Autobackup 于 01-9 月 -09

RMAN>
```

当备份整个数据库时，RMAN 将自动备份控制文件和服务器参数文件，其实这取决于 RMAN 的 CONFIGURE CONTROLFILE AUTOBACKUP 参数，可以设置该参数值为 ON 使得在使用 RMAN 执行任何备份指令时，自动备份控制文件和 SPFILE 文件，参数配置如下例所示。

```
RMAN> configure controlfile autobackup on;
```

在上例中，我们就启用了控制文件的自动备份，这样在使用 RMAN 执行任何数据备份时都自动备份控制文件。

在备份整个数据库时，如启用了快闪恢复区，则使用简单的 BAKCUP DATABASE 备份指令。

04 最后打开数据库，如下例所示。

```
RMAN> alter database open;
```

```
数据库已打开
```

此时，使用 RMAN 完成了整个数据库的脱机备份。

8.7　RMAN备份控制文件

RMAN 可以单独备份控制文件，如果没有启用快闪恢复区，则使用 FORMAT 参数指定控制文件的备份目录，如果启用了快闪恢复区，RMAN 会自动将控制文件拷贝到快闪恢复区的备份集中（BACKUPSET 目录下）。下面通过两个例子演示如何使用备份控制文件的指令和整个备份过程。

例子 8-7　在没有启用快闪恢复区时备份控制文件。

```
RMAN> backup current controlfile format
2> 'f:\pump\backup_ctl_%u.dbf';
启动 backup 于 29-8 月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
通道 ORA_DISK_1: 正在启动段 1 于 29-8 月 -09
通道 ORA_DISK_1: 已完成段 1 于 29-8 月 -09
段句柄=F:\PUMP\BACKUP_CTL_0CKNTU1G.DBF 标记=TAG20090829T171527 注释
=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 29-8 月 -09
```



在本章的备份例子中多次使用替换变量%U，它的作用是产生唯一的备份文件名。因为没有使用快闪恢复区，所以在执行控制文件恢复时，DBA 必须知道备份目录，显然这增加了 DBA 的工作负担。

下面是使用快闪恢复区时的备份控制文件方式例子。

例子 8-8　在启用快闪恢复区时备份控制文件。

```
RMAN> backup current controlfile ;

启动 backup 于 29-8 月 -09
使用通道 ORA DISK 1
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
通道 ORA DISK 1: 正在启动段 1 于 29-8 月 -09
通道 ORA DISK 1: 已完成段 1 于 29-8 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2009 08
29\O1 MF NCNNF TAG20090829T17179 59KWK66T .B
KP 标记=TAG20090829T17179 注释=NONE
```

```
通道 ORA_DISK_1: 备份集已完成，经过时间:00:00:02
完成 backup 于 29-8 月 -09
```

使用快闪恢复区的好处就是 Oracle 自动管理文件的备份目录，DBA 也不需要记住这个目录，在恢复时同样不需要记住该目录。RMAN 将使用 RMAN 信息库记录的信息找到备份的文件集。

8.8　RMAN的相关概念与配置参数

在进一步讨论使用 RMAN 进行更多类型的备份前，我们有必要说明几个 RMAN 概念，这些概念也多次出现在备份输出过程中，说明如下。

- 备份集：备份集是一个逻辑数据集合，由一个或多个RMAN的备份片组成。备份片是RMAN格式的操作系统文件，包含一个数据文件、一个控制文件或者归档日志文件。默认情况下，在执行 RMAN 的备份时，将产生备份文件的备份集，备份集只有 RMAN 可以识别，所以在恢复时必须使用 RMAN 来访问备份集实现恢复。
- 通道：RMAN 是通过与数据库服务器的会话建立连接，通道代表这个连接，它指定了备份或恢复数据库的备份集所在的设备，如磁盘或磁带。
- 映像拷贝：映像拷贝是数据库文件的操作系统文件的一个备份，就如使用操作系统的 COPY 指令备份的文件一样。使用 RMAN 将默认创建备份集，它是数据集的一个逻辑数据结构，也可以设置备份类型为 COPY，使得使用 RMAN 的任何备份不产生备份集，而产生映像拷贝，如下例所示。

例子 8-9 设置备份类型为 COPY。

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY
```

也可以使用 BACKUP AS COPY 指令实现备份数据的映像拷贝，如下面例子所示，分别实现整个数据库、单个表空间、数据文件的映像拷贝。

例子 8-10 映像拷贝整个数据库。

```
RMAN> BACKUP AS COPY DATABASE
```

例子 8-11 映像拷贝单个表空间。

```
RMAN> BACKUP AS COPY TABLESPACE USERS
```

例子 8-12 映像拷贝整个数据库一个数据文件。

```
RMAN> BACKUP AS COPY DATAFILE 3
```

参数 DATAFILE 后的数值表示数据文件 ID，它与数据字典 DBA_DATA_FILES 中的 FILE_ID 参数一致。

接下来我们分析一下 RMAN 的配置参数，首先登录 RMAN 之后，使用 SHOW ALL 指令显示当前的所有 RMAN 参数，如下例所示。

例子 8-13 查看 RMAN 的配置参数。

```
RMAN> show all;
```

```
RMAN 配置参数为:
CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default
CONFIGURE BACKUP OPTIMIZATION OFF; # default
CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default
CONFIGURE CONTROLFILE AUTOBACKUP OFF; # default
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F'; # default
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE MAXSETSIZE TO UNLIMITED; # default
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default
CONFIGURE SNAPSHOT CONTROLFILE NAME TO 'F:\ORACLE\
PRODUCT\10.2.0\DB_1\DATABASE\SNCFORCL.ORA'; # default
```

可以根据需要更改上述中的参数，我们先解释部分 RMAN 参数的含义，然后说明如何设置该参数。

- CONFIGURE RETENTION POLICY TO REDUNDANCY 1：该参数说明保留备份的副本数量，如每天都备份一个数据文件。上述参数 1 说明只保留一个该数据文件的副本。
- CONFIGURE DEFAULT DEVICE TYPE TO DISK: 该配置参数说明备份的数据文件默认备份到数据库服务器的磁盘上。该参数可以更改为备份到磁带上，如下例所示。

例子 8-14 更改 RMAN 的备份设备类型为磁带。

```
RMAN> configure default device type to sbt;

新的 RMAN 配置参数:
CONFIGURE DEFAULT DEVICE TYPE TO 'SBT_TAPE';
已成功存储新的 RMAN 配置参数
释放的通道: ORA_DISK_1
```

这里仅仅是为了说明设备类型的更改方式，读者最好使用如下指令将设备类型恢复为磁盘。

```
RMAN> configure default device type to disk;
```

- CONFIGURE BACKUP OPTIMIZATION OFF：配置备份优化模式不使用备份优化。使用备份优化的作用是如果已经备份了某个文件的相同版本，则不会再备份该文件。可以使用如下指令打开备份优化。

```
RMAN> CONFIGURE BACKUP OPTIMIZATION ON;

新的 RMAN 配置参数:
CONFIGURE BACKUP OPTIMIZATION ON;
已成功存储新的 RMAN 配置参数
```

- CONFIGURE CONTROLFILE AUTOBACKUP OFF：配置模式不启动控制文件的自动备份。修改方式就是将 OFF 设置为 ON，修改指令如下所示。

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

- CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET: 该参数说明 RMAN 在备份和恢复中运行的通道数量，在执行备份或恢复时，通道数量越多，则任务执行时间越短。修改并行数的指令如下所示。

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
```

8.9 RMAN联机备份

本节我们学习 RMAN 的联机备份。在使用联机备份前必须满足一定的条件，比如使数据库处于归档模式等，然后我们通过例子详述如何备份整个数据库、表空间以及数据文件。

8.9.1 联机备份前的准备工作

在使用 RMAN 进行联机联机备份前，必须设置快闪恢复区，将 DB_RECOVERY_FILE_DEST 参数指定的目录作为归档重做日志备份的默认位置，并且将快闪恢复区的大小设置得足够大，然后需要将 LOG_ARCHIVE_START 参数的值设置为 TRUE，如下例所示。

例子 8-15 将参数 LOG_ARCHIVE_START 设置为 TRUE。

```
SQL> alter system set log archive start = true scope =spfile;

系统已更改。
```

在进行联机备份前都要求将数据库置于归档模式，因为处于联机备份的数据库中要备份的所有数据文件头中的 SCN 被锁定。但此时在数据库中的数据文件的表仍然可以被访问，并执行 DML 操作，但是这些修改的数据不能写入数据文件，Oracle 的重做日志进程将这些变化的数据全部写到重做日志文件。如果备份的时间很长，而且在这期间产生了大量的变化数据，重做日志会切换从而将这些变化的数据写到归档日志文件中。

显然，RMAN 的联机备份使得数据库可以继续运行，而且通过 RMAN 可以备份整个数据库、一个表空间或者一个数据文件，可以灵活选择备份的粒度。对于超大型数据库，如果备份整个数据库是相当耗时的，而在生产数据库中往往只需要备份某个重要的表空间或数据文件。联机备份时启用归档模式，不会丢失数据更新。在介质故障时，可以实现数据库的全恢复。但是要注意，必须小心保存或备份归档日志，因为一旦它丢失或损坏就无法实现数据库完全恢复。

最后需要将数据库设置为归档模式。其操作步骤是先关闭数据库，再启动数据库到 MOUNT 状态，然后使用 ALTER DATABASE ARCHIVELOG 指令将数据库设置为归档模式。打开数据库就可以进行 RMAN 联机热备份了。将数据库设置为归档模式的指令如下例所示。

例子 8-16 将数据库设置为归档模式。

```
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL>
SQL> startup mount;
```

```
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  190380 bytes
Variable Size              9327928 bytes
Database Buffers         352321536 bytes
Redo Buffers               7135232 bytes
数据库装载完毕。
SQL> alter database archivelog;

数据库已更改。
```

为了验证修改结果，最好使用下例所示的指令查看当前数据库的归档模式。

例子 8-17　查看数据库的归档模式。

```
SQL> archive log list;
数据库日志模式              存档模式
自动存档                    启用
存档终点                    USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列          187
下一个存档日志序列          189
当前日志序列                189
```

下面依次通过例子说明如何使用 RMAN 联机备份整个数据库、备份一个表空间、备份一个数据文件，以及备份当前的控制文件。

8.9.2　联机备份整个数据库

本节我们演示如何使用 BACKUP DATABASE 指令联机备份整个数据库，如下例所示。

例子 8-18　使用 RMAN 备份整个数据库。

```
RMAN> backup database;

启动 backup 于 01-9 月 -09
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=141 devtype=DISK
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00002
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
输入数据文件 fno=00001
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
输入数据文件 fno=00005
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
............
通道 ORA_DISK_1: 启动全部数据文件备份集
```

```
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
在备份集中包含当前的 SPFILE
通道 ORA_DISK_1: 正在启动段 1 于 01-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_01\O1
_MF_NCSNF_TAG20090901T01934_59R229XS_.B
P 标记=TAG20090901T01934 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:03
完成 backup 于 01-9 月 -09 完成
```

此时，我们成功备份了整个数据库，备份后数据库文件名称和存储目录都由 RMAN 自己维护，不需要 DBA 干预。这极大地减少了备份和恢复中出错的几率。



在备份整个数据库时，其实就是备份了数据文件，其中包含了当前的控制文件和参数文件。而重做日志文件或归档日志文件不是联机状态数据库全备份的内容，所以使用联机热备份的数据库在数据恢复时需要 recover 数据库，即将联机备份开始到故障点之间的所有提交的数据重新写入数据文件。

8.9.3　联机备份一个表空间

在很多情况下，经常需要备份一个表空间，比如 SYSTEM 表空间，此时就可以使用 BACKUP TABLESPACE 指令联机备份一个表空间，如下例所示。

例子 8-19　使用 RMAN 备份表空间。

```
RMAN> backup tablespace sysaux;

启动 backup 于 01-9 月 -09
使用目标数据库控制文件替代恢复目录
分配的通道: ORA DISK 1
通道 ORA DISK 1: sid=142 devtype=DISK
通道 ORA DISK 1: 启动全部数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
通道 ORA DISK 1: 正在启动段 1 于 01-9 月 -09
通道 ORA DISK 1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2009 09 01\O1
_MF_NNNDF_TAG20090901T010402_59R0N2K2_.B
KP 标记=TAG20090901T010402 注释=NONE
通道 ORA DISK 1: 备份集已完成, 经过时间:00:00:9
完成 backup 于 01-9 月 -09
```

上述例子，我们备份了表空间 SYSAUX，备份的文件类型为备份集。在第一次使用 RMAN 备份生成备份集时，在 RMAN 的快闪恢复区中会自动创建一个目录 BACKUPSET，还会根据日期创建新的目录，将一天中的备份集放在一个根据日期创建的目录下。

8.9.4 联机备份一个数据文件

在备份一个数据文件前，我们先查看当前数据库中的所有数据文件，以便于选择需要备份的数据文件，如下例所示。

例子 8-20 查看当前数据库的所有数据文件。

```
SQL> col file_name for a55
SQL> set line 100
SQL> select file_id,file_name,tablespace_name
  2* from dba_data_files

   FILE_ID FILE_NAME                                               TABLESPACE_NAME
---------- --------------------------- -----------------------------------
     4 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF       USERS
     3 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF      SYSAUX
     2 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF     UNDOTBS1
     1 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF      SYSTEM
     5 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF     EXAMPLE
```

在备份数据文件时，可以使用 FILE_ID 或者文件名来标识数据文件。下面的例子用于备份表空间 SYSAUX 中的数据文件，它的 FILE_ID 为 3。

例子 8-21 使用 RMAN 备份数据文件。

```
RMAN> backup datafile 3;

启动 backup 于 01-9 月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动全部数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00003
name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
通道 ORA_DISK_1: 正在启动段 1 于 01-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\2009_09_01\O1
_MF_NNNDF_TAG20090901T010753_59R0V9XS_.B
KP 标记=TAG20090901T010753 注释=NONE
通道 ORA_DISK_1: 备份集已完成，经过时间:00:00:9
完成 backup 于 01-9 月 -09
```

使用 BACKUP CURRENT CONTROLFILE 联机备份当前控制文件，如下例所示。

例子 8-22 使用 RMAN 备份控制文件。

```
RMAN> backup current controlfile;

启动 backup 于 01-9 月 -09
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=139 devtype=DISK
通道 ORA_DISK_1: 启动全部数据文件备份集
```

```
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
通道 ORA_DISK_1: 正在启动段 1 于 01-9 月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9 月 -09
段句柄
=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2009 09 01\O1
_MF_NCNNF_TAG20090901T010441_59R0OBJM_.B
KP 标记=TAG20090901T010441 注释=NONE
通道 ORA DISK 1: 备份集已完成, 经过时间:00:00:04
完成 backup 于 01-9 月 -09
```

虽然在备份整个数据库时，RMAN 自动备份了控制文件，但是经常联机备份控制文件是个好习惯，一旦数据库结构发生了变化，如新建了表空间、添加或删除了数据文件等。

8.10 RMAN的增量备份

在使用 BACKUP DATABASE 时，都是全库备份，显然每次这样的备份很耗费时间也占用磁盘空间。而 RMAN 的增量备份具有很多优势，它只备份自上次全备份以来变化的数据。显然增量备份比全库备份要块，因为增量备份的数据量明显减少（相对于全库备份而言）。

这里需要解释两个级别的增量备份，级别 0 的增量备份和级别 1 的增量备份。其中级别 0 的增量备份与全库备份相同，而级别 1 备份是执行的是差异备份，即对级别 0 备份后变化的数据做备份，显然级别 0 备份是级别 1 备份的数据基础。级别 0 备份如下例所示。

例子 8-23 使用 RMAN 实现增量备份的级别 0 备份。

```
RMAN> backup incremental level 0 database;

启动 backup 于 01-9 月 -09
使用通道 ORA DISK 1
通道 ORA DISK 1: 启动增量级别 0 数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
输入数据文件 fno=00002 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\
UNDOTBS01.DBF
输入数据文件 fno=00001 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\
SYSTEM01.DBF
输入数据文件 fno=00003 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\SYSAUX01.DBF
输入数据文件 fno=00005 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
通道 ORA DISK 1: 正在启动段 1 于 01-9 月 -09
通道 ORA DISK 1: 已完成段 1 于 01-9 月 -09
段句柄=F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\
ORCL\BACKUPSET\2009 09 01\O1 MF NNND0 TAG20090901T171114 59SS9MDP .B
KP 标记=TAG20090901T171114 注释=NONE
通道 ORA DISK 1: 备份集已完成, 经过时间:00:02:56
通道 ORA DISK 1: 启动增量级别 0 数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
```

```
在备份集中包含当前的 SPFILE
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\
ORCL\BACKUPSET\2009_09_01\O1_MF_NCSN0_TAG20090901T171114_59SSH4CQ_.B
KP 标记=TAG20090901T171114 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:03
完成 backup 于 01-9月 -09
```

上例的全库备份集是增量备份的数据基础，在使用增量备份的级别 1 的第一次备份时，将变化的数据记录到增量备份集。而当第二次使用级别 1 增量备份时，只备份自上次增量备份以来的所有变化的数据。这种级别 1 的增量备份称为差异备份。还有一种级别 1 的增量备份叫累积备份，每次实现增量备份时，它总是备份自级别 0 备份以来所有变化的数据。显然差异增量备份会有多个备份文件，而累积增量备份只有一个备份文件，所以使用累积增量备份可以减少数据库的恢复时间。下面是使用级别 1 增量备份的例子，它采用级别 1 的差异增量备份。

例子 8-24 使用 RMAN 实现增量备份的级别 1 备份。

```
RMAN> backup incremental level 1 database;

启动 backup 于 01-9月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 启动增量级别 1 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
输入数据文件 fno=00002 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\UNDOTBS01.DBF
输入数据文件 fno=00001 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\SYSTEM01.DBF
输入数据文件 fno=00003 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\SYSAUX01.DBF
输入数据文件 fno=00005 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\
ORCL\EXAMPLE01.DBF
输入数据文件 fno=00004 name=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
略过数据文件 00004, 因为它未更改
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\
ORCL\BACKUPSET\2009_09_01\O1_MF_NNND1_TAG20090901T172008_59SST9QG_.B
KP 标记=TAG20090901T172008 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:55
通道 ORA_DISK_1: 启动增量级别 1 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集中的数据文件
备份集中包括当前控制文件
在备份集中包含当前的 SPFILE
通道 ORA_DISK_1: 正在启动段 1 于 01-9月 -09
通道 ORA_DISK_1: 已完成段 1 于 01-9月 -09
段句柄=F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\
ORCL\BACKUPSET\2009_09_01\O1_MF_NCSN1_TAG20090901T172008_59SSW2PG_.B
KP 标记=TAG20090901T172008 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:03
```

```
完成 backup 于 01-9月 -09
```

此时，我们完成了使用 RMAN 实现增量备份的所有步骤。在生产数据库中，读者可以使用操作系统工具编写脚本文件，从而实现增量备份的自动化。

8.11　快速增量备份

在 8.10 节我们介绍如何实现增量备份，使用增量备份大大减少了全库备份的时间，同时也节约了存储空间，但是使用增量备份必须扫描整个数据文件，因为无论在上次增量备份数据库是否发生变化，都要进行一次全扫描来确认是否有变化的数据。为了避免这种情况的发生，Oracle 提供了快速增量备份的方案，其原理是将数据库中发生变化的数据块位置记录在一个更改跟踪文件中，这样在下次实现增量备份时，就可以通过该文件来备份变化的数据，这样就减少了全库扫描的时间。

在启动了块更改跟踪特性后，会启动一个后台进程 CTWR 负责将变化的数据块的位置写入定义的块跟踪文件。启动快跟踪特性，如下例所示。

例子 8-25　启动快跟踪特性。

```
SQL> alter database enable block change tracking
  2  using file 'e:/oracle/product/10.2.0/oradata/chtrack.log';

数据库已更改。
```

此时我们将块跟踪文件保存在与数据文件相同的目录下，文件名为 chtrack.log。如果该文件丢失或者损坏会造成数据库无法启动，需要禁用块跟踪特性后方可成功启动数据库。

下面我们通过数据字典视图 v$block_change_tracking 查询是否启动块跟踪特性。

例子 8-26　查询是否启动块跟踪特性。

```
SQL> col filename for a50
SQL> select filename,status,bytes from v$block_change_tracking

FILENAME                                            STATUS       BYTES
--------------------------------------------------  ------------ ----------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\CHTRACK.LOG        ENABLED      11599872
```

通过视图查询清楚地表明已经启动块跟踪特性，记录数据块变化的文件存储在系统的数据文件目录中，即 E:\ORACLE\PRODUCT\10.2.0\ORADATA\CHTRACK.LOG。文件大小为 11599872 字节。

在使用块跟踪特性过程中，如果需要重命名或更改跟踪文件的存放位置，可以使用 ALTER DATABASE RENAME FILE 指令实现，但是必须将数据库启动到 MOUNT 状态，如下例所示。

例子 8-27　更改块跟踪文件的存储位置。

```
SQL> alter database rename file
  2  'e:/oracle/product/10.2.0/oradata/chtrack.log'
  3  to
  4  'e:/oracle/product/10.2.0/oradata/lszpri/chtrack.log';

数据库已更改。
```

此时，将块跟踪文件从 e:/oracle/product/10.2.0/oradata/chtrack.log 更改到新的目录下，新目录为 e:/oracle/product/10.2.0/oradata/lszpri/chtrack.log。

在不需要时，可以通过如下方式禁用块跟踪特性，如下例所示。

例子 8-28　禁用块跟踪特性。

```
SQL> alter database disable block change tracking;

数据库已更改。
```

在禁用块跟踪特性后，通过数据字典视图 v$block_change_tracking 确认禁用结果，如下例所示。

例子 8-29　查看禁用块跟踪特性结果。

```
SQL> select filename,status,bytes from v$block_change_tracking;

FILENAME                                      STATUS           BYTES
--------------------------------------------- ------------------ ----------
                                             DISABLED
```

从输出看出，当前的块跟踪特性为 DISABLED 说明禁用成功，此时的数据文件为空。

8.12　创建和维护恢复目录

恢复目录保存了 RMAN 信息库的信息。Oracle 推荐使用恢复目录保存 RMAN 信息库，在信息库中保存了数据文件备份集或映像拷贝、表空间和数据文件信息以及 RMAN 的配置信息。使用恢复目录，RMAN 在一定条件下读取目标库的控制文件来更新恢复目录中保存的关于控制文件、数据文件等信息。

下面演示在同一个数据库上创建恢复目录的步骤和方法。

01 创建恢复目录用户。创建用户用于恢复目录，为该用户创建默认表空间用来存储恢复目录，如下例所示。

例子 8-30　创建恢复目录用户。

```
SQL> connect system/oracle@orcl
已连接。
SQL> create user rman_backup identified by rman
  2  temporary tablespace temp
  3  default tablespace users
  4  quota unlimited on users
  5  ;

用户已创建。
```

以上创建了用户 rman_backup，该用户使用 users 表空间存储恢复目录。在笔者的 Oracle 数据库服务器中 USERS 表空间，同时也是数据文件的表空间。在生产数据库中最好新建表空间，并且将表空间的数据文件创建在其他磁盘上。

02 为了使用新用户，首先对新用户 rman_backup 进行授权，使得该用户成为恢复目录的拥

有者，将 RECOVERY_CATALOG_OWNER 角色赋予用户 rman_backup，如下例所示。

例子 8-31　为恢复目录用户授权。

```
SQL> grant connect,resource to rman_backup;

授权成功。

SQL> grant recovery_catalog_owner to rman_backup;

授权成功。
```

03 连接到恢复目录和目标数据库。

要使用恢复目录，必须先连接到恢复目录数据库，如下例所示。

例子 8-32　连接到恢复目录数据库。

```
D:\>rman catalog rman_backup/rman@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 08:52:53 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到恢复目录数据库
```



在笔者电脑上恢复目录数据库和目标数据库在一个数据库上，所以上述方式也同时连接到了目标数据库。如果二者不在同一数据库上，如目标数据库在 leejia 数据库上，则在连接到恢复目录数据库后，再连接到目标数据库，如下面例子所示。

例子 8-33　连接到目标数据库。

```
RMAN> connect target system/oracle@leejia
连接到目标数据库: ORCL (DBID=121982901)
```

上例中，我们先连接到恢复目录数据库，然后在 RMAN 环境下再连接到名为 LEEJIA 的目标数据库上，也可以使用如下指令一次连接到恢复目录和目标数据库。

例子 8-34　连接到恢复目录数据库和目标数据库。

```
RMAN> connect catalog rman_backup/rman@orcl target system/oracle@leejia
```

04 创建恢复目录。

在刚才创建的用户 rman_backup 中创建恢复目录，首先使用用户 rman_backup 登录数据库，然后使用 create catalog 指令创建恢复目录，如下例所示。

例子 8-35　创建恢复目录。

```
D:\>rman catalog rman_backup/rman@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 09:02:9 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.
```

```
连接到恢复目录数据库

RMAN> create catalog

恢复目录已创建
```

此时的恢复目录保存在USERS表空间中，该表空间就是在创建rman_bakcup用户时指定的默认表空间。在不需要恢复目录时，可以使用DROP CATALOG指令删除恢复目录，读者可以自己验证。

05 注册目标数据库。

在创建了恢复目录后注册目标数据库，目的是使得恢复目录知道目标数据库的名字，并自动与目标数据库通信以获得相关的元数据。要注册目标数据库，必须首先连接到目标数据库，如下例所示。

例子 8-36 在恢复目录中注册目标数据库。

```
D:\>rman catalog rman_backup/rman@orcl target system/oracl@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 09:16:56 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到目标数据库: ORCL (DBID=121982901)
连接到恢复目录数据库
RMAN> register database;

注册在恢复目录中的数据库
正在启动全部恢复目录的 resync
完成全部 resync
```

在上例中把目标数据库orcl成功注册到恢复目录中，“正在启动全部恢复目录的resync”的含义是RMAN读取目标库的控制文件信息来保存关于数据文件、日志切换等的元数据信息，因为在刚注册时恢复目录没有任何关于目标数据库的注册信息，所以使用同步的方式获取所需信息。其实，在目标数据库结构发生变化后，可以使用手工同步的方式，如下例所示。

例子 8-37 同步恢复目录。

```
RMAN> resync catalog;

正在启动全部恢复目录的 resync
完成全部 resync
```

注册成功后，我们可以查看目标数据库的关于数据文件的信息，这些信息是RMAN通过恢复目录读取的，如图8-2所示。



图 8-2　查看目标库的数据文件信息。

8.13　RMAN的脚本管理

对于长指令的 RMAN 备份操作，Oracle 提供了脚本语言功能，使得用户正对特定的任务编写备份脚本，然后将脚本存储在恢复目录或存储为文本文件。下面通过例子说明如何创建和使用脚本。

创建 RMAN 备份脚本的指令，如下所示。

例子 8-38 创建 RMAN 备份脚本。

```
D:\>rman catalog rman_backup/rman@orcl target system/oracle@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9月 3 09:33:07 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到目标数据库: ORCL (DBID=121982901)
连接到恢复目录数据库

RMAN> create script rman backup{
2> sql 'alter system checkpoint';
3> backup database format
4> 'f:\offline backup\back %u.dbf';
5> backup current controlfile format
6> 'f:\offline backup\back ctl %u.dbf';
7> }

已创建脚本 rman backup
```

注意在创建 RMAN 备份脚本时，必须连接到恢复目录和目的数据库，否则不能创建成功。

执行 RMAN 脚本，此时使用 RUN 指令和 EXECUTE SCRIPT 指令执行创建的脚本。其实执行脚本的过程就是执行一系列的 SQL 语句的过程，RMAN 脚本类似于 WINDOWS 中的批处理文件，如下例所示。

例子 8-39 执行脚本。

```
RMAN> run {execute script rman backup;}

正在执行脚本: rman backup
sql 语句: alter system checkpoint
启动 backup 于 03-9 月 -09
………
```

在介绍完了如何创建脚本以及执行脚本，下面我们介绍如何使用操作系统文件存储 RMAN 指令，并在 RMAN 中直接调用该文件执行 RMAN 命令。首先创建一个 rman_backup.rcv 文件，如图 8-3 所示。

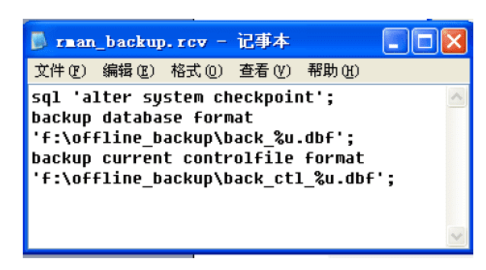


图 8-3 创建执行 RMAN 指令的操作系统文件

然后将该文件保存在 D 盘的根目录下，在 RMAN 中调用操作系统文件执行 RMAN 指令，如下例所示。

例子 8-40 调用操作系统文件执行 RMAN 指令。

```
D:\>rman catalog rman backup/rman@orcl target system/oracle@orcl cmdfile
'rman backup.rcv'

恢复管理器: Release 11.1.0.6.0 - Production on 星期四 9 月 3 10:05:45 2009

Copyright (c) 1982, 2007, Oracle. All rights reserved.

连接到目标数据库: ORCL (DBID=121982901)
连接到恢复目录数据库

RMAN> sql 'alter system checkpoint';
2> backup database format
3> 'f:\offline_backup\back_%u.dbf';
4> backup current controlfile format
5> 'f:\offline_backup\back_ctl_%u.dbf';
6>
7>
sql 语句: alter system checkpoint

启动 backup 于 03-9 月 -09
分配的通道: ORA_DISK_1
…………
完成 Control File and SPFILE Autobackup 于 03-9 月 -09
恢复管理器完成。
```

为了编辑方便，也可以将存储在恢复目录中的脚本文件转换为操作系统文件，如下例所示。

例子 8-41 将脚本文件转换为操作系统文件。

```
RMAN> print script rman_backup to file 'rman_backup.txt';

已将脚本 rman_backup 写入文件 rman_backup.txt
```

8.14 使用RMAN实现脱机备份的恢复（NOARCHIVELOG模式）

我们以恢复整个数据库为例，说明在非归档模式下使用 RMAN 实现脱机备份的恢复。此时，自脱机备份以来变化的数据可能部分丢失。读者应该还有印象，联机重做日志文件是循环使用的，一旦写满一个日志文件将切换到下一个，新的循环开始将覆盖掉部分变化的数据。这样的恢复其实是不完全恢复，因为数据库工作在非归档模式下。下面给出具体的步骤说明这种情况下如何恢复整个数据库。

01 把数据库启动到 NOMOUNT 状态，下面例子假设数据库在运行，所以先关闭数据库，然后使用 NOMOUNT 参数启动数据库。

例子 8-42 将数据库启动到 NOMOUNT 状态。

```
RMAN> startup nomount;

已连接到目标数据库 (未启动)
Oracle 实例已启动

系统全局区域总计      603979776 字节

Fixed Size                   190380 字节
Variable Size              218106804 字节
Database Buffers           377487360 字节
Redo Buffers                 7135232 字节
```

02 从备份的控制文件中恢复控制文件，如下例所示。

例子 8-43 恢复控制文件。

```
RMAN> restore controlfile from autobackup;

启动 restore 于 01-9月 -09
分配的通道: ORA DISK 1
通道 ORA DISK 1: sid=157 devtype=DISK

恢复区域目标: F:\oracle\product\10.2.0/flash recovery area
用于搜索的数据库名 (或数据库的唯一名称): ORCL
通道 ORA DISK 1: 在恢复区域中找到自动备份
通道 ORA DISK 1: 已找到的自动备份:
F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\AUTOBACKUP\2009_09_01\O
```

```
1_MF_S_69646
5171_59TDD3LK_.BKP
通道 ORA_DISK_1: 从自动备份复原控制文件已完成
输出文件名=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL01.CTL
输出文件名=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL02.CTL
输出文件名=F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL03.CTL
完成 restore 于 01-9 月 -09
```

在上例中，要求使用自动备份的控制文件来恢复当前的控制文件。在使用 RMAN 实现脱机备份时，我们已经将 RMAN 的 CONFIGURE CONTROLFILE AUTOBACKUP 设置为 ON，所以在数据库备份时已经自动备份了控制文件，因此上例的执行可以成功。

03 将数据库切换到 MOUNT 状态，如下例所示。

例子 8-44　数据库切换到 MOUNT 状态。

```
RMAN> alter database mount;

数据库已装载
释放的通道: ORA_DISK_1
```

此时，打开了控制文件，但是这个控制文件是从控制文件的自动备份中恢复的。接下来使用脱机备份集重建数据库。

04 重建数据库，如下例所示。

例子 8-45　重建数据库。

```
MAN> restore database ;

启动 restore 于 01-9 月 -09
使用通道 ORA_DISK_1

通道 ORA_DISK_1: 正在开始恢复数据文件备份集
通道 ORA_DISK_1: 正在指定从备份集恢复的数据文件
正将数据文件 00001 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
正将数据文件 00003 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
…………
通道 ORA_DISK_1: 恢复完成, 用时: 00:01:9
完成 restore 于 01-9 月 -09
```

05 打开数据完成数据库恢复，如下例所示。

例子 8-46　完成数据库恢复。

```
RMAN> alter database open;

数据库已打开
```

8.15　使用RMAN实现脱机备份的恢复（ARCHIVELOG模式）

在归档模式下，利用 RMAN 的脱机备份、所有归档重做日志，以及当前的重做日志文件可以实现数据库的完全恢复。要求在使用 RMAN 脱机备份以来数据库一直运行在归档模式，且归档文件以及重做日志文件没有损坏。

我们以恢复整个数据库为例说明，当数据库自备份处于归档模式时，如何使用 RMAN 实现数据库的完全恢复。其步骤说明如下。

01 如果数据库在运行，则使用 SHUTDOWN IMMEDIATE 指令关闭数据库，如下例所示。

例子 8-47 在 RMAN 中关闭数据库并启动到 MOUNT 状态。

```
RMAN> startup mount;

已连接到目标数据库 (未启动)
Oracle 实例已启动
数据库已装载

系统全局区域总计     603979776 字节

Fixed Size                  190380 字节
Variable Size              98438452 字节
Database Buffers           327155712 字节
Redo Buffers                7135232 字节
```

02 此时需要重建（RESTORE）数据库。从最近备份的全库备份集中重建整个数据库，其过程是将备份集中的数据文件拷贝到它原来的目录下，但是备份集的数据格式只有 RMAN 可以识别。方法如下例所示。

例子 8-48 使用 RMAN 重建数据库。

```
RMAN> restore database;

启动 restore 于 01-9月 -09
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=156 devtype=DISK

通道 ORA_DISK_1: 正在开始恢复数据文件备份集
通道 ORA_DISK_1: 正在指定从备份集恢复的数据文件
正将数据文件 00001 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
正将数据文件 00002 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
正将数据文件 00003 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
正将数据文件 00004 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
正将数据文件 00005 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
………..
通道 ORA_DISK_1: 恢复完成, 用时: 00:02:15
完成 restore 于 01-9月 -09
```

03 恢复（RECOVER）数据库，如下例所示。

例子 8-49　恢复数据库。

```
RMAN> recover database;

启动 recover 于 01-9 月 -09
使用通道 ORA_DISK_1
通道 ORA_DISK_1: 正在开始恢复增量数据文件备份集
通道 ORA_DISK_1: 正在指定从备份集恢复的数据文件
..........
正在开始介质的恢复
介质恢复完成, 用时: 00:00:06

完成 recover 于 01-9 月 -09
```



使用 RECOVER DATABASE 的目的是将在重建数据库时使用的备份集以来所有的用户提交数据重写入数据文件，完成数据库的完全恢复。此时，RMAN 会根据恢复过程应用相应的备份集或者归档日志文件。

04 打开数据库完成全库的完全恢复，如下例所示。

例子 8-50　使用 RMAN 将数据库从 MOUNT 状态切换到 OPEN 状态。

```
RMAN> alter database open;

数据库已打开
```

8.16　从联机热备份使用RMAN恢复

RMAN 支持从联机热备份中恢复整个数据库、表空间，以及数据文件。恢复整个数据的其方式与使用 RMAN 从脱机冷备份恢复（ARCHIVELOG 模式）步骤相同。如果是恢复表空间或者仅仅恢复一个数据文件，步骤基本相同，但是重建和恢复这些数据库对象略有区别。下面分别介绍如何恢复一个表空间，以及如何恢复一个数据文件。

1．从 RMAN 联机备份中恢复 SYSAUX 表空间

01 启动 RMAN 并建立与数据库服务器的连接，如下例所示。

例子 8-51　启动 RMAN。

```
D:\>rman target system/oracle@orcl

恢复管理器: Release 11.1.0.6.0 - Production on 星期二 9 月 1 23:09:23 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到目标数据库: ORCL (DBID=121982901)
```

02 将需要恢复的表空间脱机，如下例所示。

例子 8-52　将表空间脱机。

```
RMAN> sql 'alter tablespace sysaux offline';

使用目标数据库控制文件替代恢复目录
sql 语句: alter tablespace sysaux offline
```

03 重建表空间 SYSAUX，使用 RESTORE TABLESPACE 指令从最近备份的与该表空间相关的文件中重建表空间，这个过程如找到最近备份集，以及备份集中与需重建表空间相关的数据文件等都由 RMAN 自动完成，如下例所示。

例子 8-53　重建表空间 SYSAUX。

```
RMAN> restore tablespace sysaux;

启动 restore 于 01-9 月 -09
分配的通道: ORA DISK 1
通道 ORA DISK 1: sid=144 devtype=DISK

通道 ORA DISK 1: 正在开始恢复数据文件备份集
通道 ORA DISK 1: 正在指定从备份集恢复的数据文件
正将数据文件 00003 恢复到 F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
通道 ORA DISK 1: 正在读取备份段 F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\
ORCL\BACKUPSET\2009 09 01\O1 MF NNNDF TAG20
090901T223404 59TD6X2G .BKP
通道 ORA_DISK_1: 已恢复备份段 1
段句柄 = F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\
ORCL\BACKUPSET\2009 09 01\O1 MF NNNDF TAG20090901T223404 59TD6X2G
.BKP 标记 = TAG20090901T223404
通道 ORA DISK 1: 恢复完成, 用时: 00:00:55
完成 restore 于 01-9 月 -09
```

04 恢复表空间，此时使用 RECOVER TABLESPACE 指令，因为备份集中表空间 SYSAUX 中的数据文件不是最新的，使用 RECOVER 将该表空间中变化的数据重新写入，如下例所示。

例子 8-54　恢复表空间。

```
RMAN> recover tablespace sysaux;

启动 recover 于 01-9 月 -09
使用通道 ORA_DISK_1

正在开始介质的恢复
介质恢复完成, 用时: 00:00:03

完成 recover 于 01-9 月 -09
```

05 将表空间联机，如下例所示。

例子 8-55　把表空间 SYSAUX 联机。

```
RMAN> sql 'alter tablespace sysaux online';

sql 语句: alter tablespace sysaux online
```

至此，完成了对表空间 SYSAUX 的基于联机 RMAN 备份的恢复。

2．使用 RMAN 恢复数据文件。

恢复数据文件的过程和恢复表空间的过程一样，我们不给出具体的例子，只给出每一步的指令，读者可以当做作业完成。

01　启动 RMAN。

```
D:\rman target system/oracle@orcl
```

02　将要恢复的数据文件脱机。

```
RMAN>sql 'alter datafile f:\oracle\product\10.2.0\oradata\orcl\sysaux01.dbf offline';
```

03　重建数据文件。

```
RMAN>restore datafile 'f:\oracle\product\10.2.0\oradata\orcl\sysaux01.dbf';
```

04　恢复数据文件。

```
RMAN>recover datafile 'f:\oracle\product\10.2.0\oradata\orcl\sysaux01.dbf';
```

05　将数据文件联机。

```
RMAN> sql 'alter datafile f:\oracle\product\10.2.0\oradata\orcl\sysaux01.dbf online';
```

8.17　RMAN实现数据块恢复

使用 RMAN 可以实现数据块级的数据恢复。在传统恢复手段中，某个数据文件的一个数据块被损坏，就造成整个数据文件无法使用，此时必须通过备份恢复整个数据文件，显然这样的方法恢复时间较长。而 RMAN 实现块级恢复，如果某个数据文件的数据块损坏，通过数据文件的完整备份就可以恢复数据块。下面我们演示如何使用 RMAN 恢复数据块。

01　我们首先需要打开 RMAN 然后备份整个数据库，这是一个全备份，如下面例子所示。

例子 8-56　打开 RMAN。

```
D:\>rman target /

恢复管理器: Release 10.2.0.1.0 - Production on 星期日 7月 11 16:04:57 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

连接到目标数据库: ORCL (DBID=1251847545)
```

例子 8-57　备份整个数据库。

```
RMAN> backup database plus archivelog;

启动 backup 于 11-7月 -10
当前日志已存档
```

```
    分配的通道：ORA_DISK_1
    通道 ORA_DISK_1: sid=136 devtype=DISK
    通道 ORA_DISK_1: 正在启动存档日志备份集
    通道 ORA_DISK_1: 正在指定备份集中的存档日志
    输入存档日志线程 =1 序列 =4 记录 ID=1 时间戳=724089956
    通道 ORA DISK 1: 正在启动段 1 于 11-7月 -10
    通道 ORA_DISK_1: 已完成段 1 于 11-7月 -10
    段句柄
    =E:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2010 07 11\O
    1_MF_ANNNN_TAG20100711T160557_63LYV6NJ_.BKP 标 记 =TAG20100711T160557 注 释
=NONE
    通道 ORA DISK 1: 备份集已完成，经过时间:00:00:02
    完成 backup 于 11-7月 -10

    启动 backup 于 11-7月 -10
    ………..
    ………..
    通道 ORA DISK 1: 备份集已完成，经过时间:00:00:02
    完成 backup 于 11-7月 -10
```

此时，我们将归档日志也放在备份集中，这不是必须的，读者可以自行选择，备份文件的存储目录我们使用 Oracle 默认的快闪恢复区。

02 接着就是做些破坏性行为，我们先关闭数据库，如下例所示。

例子 8-58　关闭数据库并使用 ULTRAEDIT 修改数据文件 RMAN.DBF。

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

数据库关闭后，使用 ULTRAEDIT 编辑数据文件 RMAN.DBF，破坏一些数据，并保存以模拟数据文件的损坏。如果我们再启动数据库就会报错，如下例所示。

例子 8-59　数据文件 6 损坏后启动数据库。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             176163764 bytes
Database Buffers          419430400 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01113: 文件 6 需要介质恢复
ORA-01110: 数据文件 6: 'D:\RMAN.DBF'
```

03 进行数据文件 6 的有效性检验。如果该数据文件中有坏块，则在数据字典视图 v$database_block_corruption 有详细记录，如下例所示。

例子 8-60 对数据文件 6 进行有效检验。

```
RMAN> backup validate datafile 6;

启动 backup 于 11-7月 -10
使用目标数据库控制文件替代恢复目录
分配的通道: ORA DISK 1
通道 ORA DISK 1: sid=155 devtype=DISK
通道 ORA DISK 1: 启动全部数据文件备份集
通道 ORA DISK 1: 正在指定备份集中的数据文件
输入数据文件 fno=00006 name=D:\RMAN.DBF
通道 ORA DISK 1: 备份集已完成, 经过时间:00:00:02
完成 backup 于 11-7月 -10
```

然后，我们通过数据字典 v$database_block_corruption 来查看数据文件 6 中损坏的数据块，如下例所示。

例子 8-61 查看数据文件 6 中损坏的数据块。

```
SQL> select *
  2  from v$database_block_corruption;

     FILE#     BLOCK#     BLOCKS CORRUPTION_CHANGE# CORRUPTIO
---------- ---------- ---------- ------------------ ---------
         6        118          1                  0 CHECKSUM
```

从输出看出数据文件 6 的数据块 118 被损坏，所以需要修复。也可以查看告警日志文件，日志文件中记录了数据文件 6 的坏块信息。如下例所示。

例子 8-62 查看告警日志文件。

```
Hex dump of (file 6, block 118) in trace file
e:\oracle\product\10.2.0\admin\orcl\udump\orcl_ora_2872.trc
Corrupt block relative dba: 0x01800076 (file 6, block 118)
Bad check value found during backing up datafile
Data in bad block:
 type: 0 format: 2 rdba: 0x00000076
 last change scn: 0x0000.00000000 seq: 0x1 flg: 0x05
 spare1: 0x0 spare2: 0x0 spare3: 0x0
 consistency value in tail: 0x00000001
 check value in block header: 0xa75f
 computed block checksum: 0x29
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
Reread of blocknum=118, file=D:\RMAN.DBF. found same corrupt data
```

无论通过数据字典视图还是告警日志文件，我们都可以确定数据文件 6 的 118 号数据块损坏。

04 我们通过 RMAN 恢复这个数据块，如下例所示。

例子 8-63 使用 RMAN 完成数据块恢复。

```
RMAN> blockrecover datafile 6 block 118 from backupset;
```

```
启动 blockrecover 于 11-7 月 -10
使用通道 ORA_DISK_1

通道 ORA_DISK_1: 正在恢复块
通道 ORA DISK 1: 正在指定要从备份集恢复的块
正在恢复数据文件 00006 的块
通道 ORA_DISK_1: 正在读取备份段
E:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2010 07 11\O1
MF_NNNDF_TAG20
100711T160559 63LYV8J3 .BKP
通道 ORA DISK 1: 已从备份段 1 恢复块
段句柄 =
E:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\BACKUPSET\2010 07 11\O1
MF NNNDF TAG20100711T160559 63LYV8J3
.BKP 标记 = TAG20100711T160559
通道 ORA DISK 1: 块恢复完成, 用时: 00:00:02

正在开始介质的恢复
某些块未恢复: 有关详细信息, 请参阅跟踪文件
介质恢复完成, 用时: 00:00:01

完成 blockrecover 于 11-7 月 -10
```

此时，提示已经成功恢复数据块。但是由于某些块未恢复，所以数据文件验证失败，可以通过数据告警日志看到这个提示。我们查看告警日志文件了解这个恢复过程，如下例所示。

例子 8-64 查看告警日志文件。

```
Starting block media recovery
Sun Jul 11 16:40:27 2010
Recovery of Online Redo Log: Thread 1 Group 1 Seq 5 Reading mem 0
  Mem# 0 errs 0: E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG
Sun Jul 11 16:40:27 2010
Recovery of Online Redo Log: Thread 1 Group 2 Seq 6 Reading mem 0
  Mem# 0 errs 0: E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO02.LOG
Sun Jul 11 16:40:27 2010
Errors in file e:\oracle\product\10.2.0\admin\orcl\udump\orcl_ora_2872.trc:
ORA-01122: 数据库文件 6 验证失败
ORA-01110: 数据文件 6: 'D:\RMAN.DBF'
ORA-01208: 数据文件是旧的版本 - 不能访问当前版本

Sun Jul 11 16:40:27 2010
Completed block media recovery
```

此时提示数据文件验证失败，所以在打开数据库前必须 RECOVER 数据文件 6 ，如下例所示。

例子 8-65 恢复数据文件 6。

```
SQL> recover datafile 6;
完成介质恢复。
```

然后，打开数据库并验证表 RMAN_EMP 是否存在，如下例所示。

例子 8-66　打开数据库并验证表 RMAN_EMP 是否恢复。

```
SQL> alter database open;

数据库已更改。
SQL> select count(ename)
  2  from rman emp;

COUNT(ENAME)
------------
          14
```

此时，我们成功打开了数据库并且通过查询验证了数据文件 6 中唯一的表已经恢复，没有任何数据丢失。

8.18　RMAN的备份恢复验证指令

本节我们介绍几个常用的 RMAN 验证指令，分别是 VALIDATE BACKUPSET、RESTORE…VALIDATE 和 RESTORE…PREVIEW。下面分别详细介绍这三个指令的作用以及用法。

8.18.1　RMAN 的 VALIDATE BACKUPSET 指令

在使用 RMAN 备份了数据库后，需要恢复时最好使用 VALIDATE BACKUPSET 指令验证备份文件的可用性，如备份的数据文件都以备份集的形式存在，在使用 VALIDATE BACKUPSET 验证备份集时，RMAN 会自动找到你指定的备份集，如下例所示。

例子 8-67　使用 VALIDATE BACKUPSET 指令验证备份集的可用性。

```
RMAN> validate backupset 30;

分配的通道: ORA DISK 1
通道 ORA DISK 1: sid=140 devtype=DISK
通道 ORA DISK 1: 正在启动数据文件备份集验证
通道 ORA DISK 1: 正在读取备份段
F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\AUTOBACKUP\2009 09 01\O
1 MF S 69646757
3 59TGHPK2 .BKP
通道 ORA DISK 1: 已恢复备份段 1
段句柄 =
F:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\AUTOBACKUP\2009 09 01\O
1 MF S 696467573 59TGHPK2 .BKP 标记 =
TAG20090901T23193
通道 ORA_DISK_1: 验证完成, 用时: 00:00:02
```

在上例中，RMAN 确实启动了数据文件备份集验证，而且“验证完成”说明此备份集是有效的，可用于恢复操作。

读者应该会问数字 30 代表什么，其实它代表了 RMAN 的所有备份集中代表某个备份集的关键字，使用如下 LIST BACKUP SUMMARY 指令查看备份集的汇总信息。

例子 8-68　查看备份集汇总信息。

```
RMAN> list backup summary;

备份列表
===============
关键字 TY  LV  S 设备类型      完成时间       段数 副本数  压缩标记
---------- --- --- -- --------------- ----------- ------ ------ -------
1        B  F  A DISK        22-8 月 -09 1     1   NO      TAG20090822T21595
…………
30       B  F  A DISK        01-9 月 -09 1     1   NO      TAG20090901T23193
31       B  F  A DISK        01-9 月 -09 1     1   NO     TAG20090901T23948
```



使用 VALIDATE BACKUPSET 只是验证了备份集中某个备份集的有效性，但是如果要知道某个表空间或数据文件是否在备份集中又该如何处理呢？ Oracle 提供了 RESTORE…VALIDATE 指令。

8.18.2　RMAN 的 RESTORE…VALIDATE 指令

RMAN 支持使用 RESTORE…VALIDATE 验证数据库对象是否在当前的备份集中，这样在用户恢复一个数据文件或一个表空间时，可以首先确认该对象备份信息是否存在，如下例所示。

例子 8-69　验证表空间 SYSAUX 备份信息是否在备份集中。

```
RMAN> restore tablespace sysaux validate;

启动 restore 于 02-9 月 -09
使用通道 ORA_DISK_1

通道 ORA DISK 1: 正在启动数据文件备份集验证
通道 ORA DISK 1: 正在读取备份段
…………
通道 ORA DISK 1: 验证完成, 用时: 00:00:40
完成 restore 于 02-9 月 -09
```

下面再给出验证一个数据文件是否在备份集中的例子，如下例所示。

例子 8-70　验证数据文件是否在备份集中。

```
RMAN> restore datafile 'F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF'
validate;

启动 restore 于 02-9 月 -09
使用通道 ORA DISK 1

通道 ORA DISK 1: 正在启动数据文件备份集验证
通道 ORA DISK 1: 正在读取备份段
…………
通道 ORA DISK 1: 验证完成, 用时: 00:00:47
完成 restore 于 02-9 月 -09
```

上述输出中的“验证完成”说明该数据文件存在于备份集中。

8.18.3 RMAN 的 RESTORE…PREVIEW 指令

用户在备份数据库前，或许想知道执行恢复的所有文件是否存在，如当恢复表空间时，想知道该表空间中的所有数据文件是否在备份集中、在恢复全库时刻的数据文件归档日志文件是否存在等等。RMAN 提供了 RESTORE……PREVIEW 指令完成这项功能，如下例所示。

例子 8-71 查看恢复整个数据库所需的备份文件是否存在。

```
RMAN> restore database preview;

启动 restore 于 02-9月 -09
使用通道 ORA_DISK_1
备份集列表
===================
…………
  备份集 27 中的数据文件列表
  文件 LV 类型 Ckp SCN    Ckp 时间    名称
  ---- -- ---- ---------- ---------- ----
  1       Full 4921182    01-9月 -09
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
  ……………
  5       Full 4921182    01-9月 -09
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
已存档的日志副本列表
关键字      Thrd Seq      S 短时间      名称
------- ---- ------- - ---------- ----
…………
16      1    1      A 01-9月 -09
F:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\ARCHIVELOG\2009_09_02\O
1 MF 1 1 59VF
OKQG .ARC
介质恢复启动 SCN 是 4921182
完成 restore 于 02-9月 -09
```

最后显示的“完成 restore”说明数据库所需要的备份文件都存在。同样可以验证恢复某个表空间或数据文件所需的备份文件是否存在，如下例所示。

例子 8-72 查看恢复表空间或数据文件所需的备份文件是否存在。

```
RMAN> restore tablespace sysaux preview;
RMAN> restore datafile 5 preview;
```

8.19 本章小结

本章介绍了 RMAN 备份与恢复。RMAN 实现了数据库备份与恢复的自动化处理，并且具有支持增量备份、指令简洁、维护方便的优点。本章重点是理解 RMAN 备份与恢复技术，掌握使用 RMAN 实现联机备份和脱机备份及其相应的恢复方法。理解归档模式和非归档模式下，数据库恢复的本质区别。RMAN 还提供了一系列的备份验证工具，使得在恢复前及时了解备份文件的状态。

第 9 章

EXP/IMP及数据库备份与恢复

数据库备份是 DBA 的一项重要日常任务。没有备份就没有恢复，所以 DBA 需要选择良好的备份方案、合适的备份工具，以及相应的恢复方案。EXP/IMP 是 Oracle 比较传统的数据库逻辑备份工具，它能实现全库或表空间的逻辑备份，但是它不支持用户的交互模式，即在备份过程中无法控制或切换备份进程，而 Oralce 的数据泵技术可以很好的实现用户交互，支持网络操作以及重启失败的备份作业。

9.1 关于备份的几个概念

或许读者经常听到以下的说法，如逻辑备份和物理备份、一致备份和非一致备份、脱机备份和联机备份、备份粒度等。读者很好地理解并把握这些概念，对于完成备份和恢复很重要。

1．逻辑备份和物理备份

逻辑备份导出数据库的结构以及数据，这些结构包括表的定义、触发器以及存储过程等数据库对象，当使用数据泵技术或 EXP/IMP 技术时实现的是逻辑备份。物理备份是将数据库的数据文件、控制文件和归档日志文件等重要文件拷贝到操作系统的其他磁盘，此时的文件保持原文件类型。

2．脱机备份和联机备份

脱机备份指在数据库关闭的情况下实现数据备份，也称为冷备份。而联机备份是数据库运行时进行的数据备份，也叫做热备份。采用联机备份还是脱机备份依赖于业务的需求，对于 7×24 小时运行的数据库显然不能使用脱机备份。但是联机备份相对复杂，必须考虑数据库的归档模式，以及设计合理的联机备份方案。

3．一致备份和非一致备份

先解释一下“恢复”（RECOVER）的概念，因为数据文件和控制文件中的系统 SCN 不一致，恢复进程必须使用归档日志文件和联机重做日志文件的数据更新数据文件中的内容，也就是将重做日志文件中用户提交的数据重新写入数据文件。再解释“一致”的概念，Oracle 为每个事务设置了一个唯一的 SCN（系统更改号），当每次事务提交时都自动增加 SCN 号，这个号码永远是唯一的。当 DBWR 写进程运行时，将触发一个检验点事件，把数据库缓冲区中所有已经提交的数据写入磁盘，并使得所有数据文件和控制文件中的 SCN 相一致。这里一致的概念就是所有数据文件和控制文件中的 SCN 相同。

而当 LGWR 将数据库缓冲区中变化的数据写入重做日志文件时，对于用户提交了数据的事务的 SCN 将记录在控制文件中，注意此时的数据文件中的 SCN 没有变化，这就叫做不一致状态。

所以一致备份和不一致备份的区别就是是否需要恢复。要实现一致备份可以关闭数据库使用脱机备份的方式，也可以使数据库处于 MOUNT 状态，使用 RMAN 工具实现。

在 7×24 小时运行的数据库中，不一致备份是唯一的选择，它并不代表这样的备份是不可靠的，只是在数据恢复时需要一个“一致”的过程。只要数据库处于归档模式，且重做日志归档文件没有损坏，就可以使用不一致的备份实现数据库的完全恢复，不会造成数据的丢失。

9.2　使用EXP指令实现逻辑备份

EXP 和 IMP 是 Oracle 比较“古老”的数据备份和恢复方式，使用 EXP 实用程序可以导出整个数据库，一个用户的所有对象，一个表空间或特定的表。使用 EXP 实用程序导出的数据必须使用 IMP 实用程序恢复备份的数据。本节讲述 EXP 备份数据的方法，以及给出备份不同数据库对象的实例，如备份整个数据库、备份特定的表空间和部分表。

9.2.1　EXP 指令详解

Oracle 的 EXP 实用程序使用命令行方式备份数据，所以它支持一系列的操作指令，通过这些指令告诉数据库要备份的数据类型、登录数据库的用户名和密码，以及其他和导出数据相关的关键字。如果用户在 WINDOWS 平台上，可以在 DOS 窗口中输入 EXP HELP = Y，然后回车，查看 EXP 的数据导出指令，如下例所示。

例子 9-1 EXP 实用程序的参数指令。

```
C:\Documents and Settings\Administrator>exp help=y

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

通过输入 EXP 命令和您的用户名/口令，导出操作将提示您输入参数：

     例如：EXP SCOTT/TIGER

或者，您也可以通过输入跟有各种参数的 EXP 命令来控制导出的运行方式。要指定参数，
您可以使用关键字：

     格式：EXP KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)
     例如：EXP SCOTT/TIGER GRANTS=Y TABLES=(EMP,DEPT,MGR) 或
           TABLES=(T1:P1,T1:P2)，如果 T1 是分区表 USERID 必须是命令行中的第一个参数。

关键字          说明 （默认值）            关键字           说明 （默认值）
-------------------------------------------------------------------
USERID          用户名/口令                FULL             导出整个文件 （N）
BUFFER          数据缓冲区大小              OWNER            所有者用户名列表
FILE            输出文件 （EXPDAT.DMP）     TABLES           表名列表
```

```
COMPRESS      导入到一个区 (Y)           RECORDLENGTH      IO 记录的长度
GRANTS        导出权限 (Y)               INCTYPE           增量导出类型
INDEXES       导出索引 (Y)               RECORD            跟踪增量导出 (Y)
DIRECT        直接路径 (N)               TRIGGERS          导出触发器 (Y)
LOG           屏幕输出的日志文件         STATISTICS        分析对象 (ESTIMATE)
ROWS          导出数据行 (Y)             PARFILE           参数文件名
CONSISTENT    交叉表的一致性 (N)         CONSTRAINTS       导出的约束条件 (Y)

OBJECT CONSISTENT         只在对象导出期间设置为只读的事务处理 (N)
FEEDBACK                  每 x 行显示进度 (0)
FILESIZE                  每个转储文件的最大大小
FLASHBACK SCN             用于将会话快照设置回以前状态的 SCN
FLASHBACK TIME            用于获取最接近指定时间的 SCN 的时间
QUERY                     用于导出表的子集的 select 子句
RESUMABLE                 遇到与空格相关的错误时挂起 (N)
RESUMABLE NAME            用于标识可恢复语句的文本字符串
RESUMABLE TIMEOUT         RESUMABLE 的等待时间
TTS FULL CHECK            对 TTS 执行完整或部分相关性检查
TABLESPACES               要导出的表空间列表
TRANSPORT TABLESPACE      导出可传输的表空间元数据 (N)
TEMPLATE                  调用 iAS 模式导出的模板名

成功终止导出，没有出现警告。
```

上述输出是在 Oracle 10g 数据库中的 EXP 参数指令。虽然在 Oracle 10g 中建议使用 Oracle 的数据泵导入、导出数据库技术，但是在 Oracle 10g 版本中仍然兼容支持 EXP 实用程序，只是它和 Oracle 9i 中的参数指令略有不同。下面我们解释部分的参数指令。

- USERID：该参数无默认值，说明登录数据库的用户名/密码。
- BUFFER：指定数据缓冲区的大小，该参数依赖于特定的操作系统，用户可以根据导出数据的性质如 LOB 数据类型，可以适当设置较大的 BUFFER 参数值。
- FILE：说明将备份的数据文件重新命名，该输出文件名默认为 EXPDAT.DMP 或者是 expdat.dmp。
- COMPRESS：该参数的默认值为 Y 或 N，使 Oracle 对输入文件进行配置，使得当引入并且重新创建对象时，对象初始化的大小为已经导出的对象大小。
- GRANTS：该参数值为 Y 或 N，用来控制授权的导出。
- INDEXES：该参数值为 Y 或 N，用来控制索引的导出。
- LOG：该参数没有默认值，说明在导出备份时是否需要创建一个备份日志，该日志记录整个备份过程。
- FULL：说明是否导出整个数据库的所有对象，该参数值为 N 或 Y。如果执行整个数据库的导出，则连接的用户必须具有 DBA 权限或者具有 EXP_FULL_DATABASE 权限。
- OWNER：该参数没有默认值，OWNER 参数说明导出特定用户的数据库对象。可以有多个用户名，使用逗号隔开。
- TABLES：说明要导出的表的名称，如果该表是属于当前连接用户，则直接输入表名，否则输入模式名.表名。此处可以使用多个表名，使用逗号隔开。
- TRIGGERS：该参数的默认值为 Y 或 N，说明是否导出该用户模式下的触发器对象。

- STATISTICS: 该参数的值为 estimate、computer 或者 none，指出产生的对象统计量的方式，如果选择了 ESTIMATE、COMPUTER 则以该方式计算统计量，如果选择了 NONE 则不使用统计量。
- CONSTRAINTS: 该参数的值为 Y 或 N，说明是否要导出约束。
- FEEDBACK: 每隔一定的行数显示备份进展情况，该参数的值从 0 到任何有效的数字。
- FILESIZE: 指出每个转储文件的最大值。
- TABLESPACES: 说明要导出的表空间的名称，使得 EXP 程序可以从这些表空间中导出数据。该参数可以使用多个表空间名，用逗号隔开。
- RESUMABLE: 该参数的值为 Y 或 N，指出 EXP 是否使用 Oracle 的可恢复空间管理工具，使用该工具导出数据时，如发生与空间相关的错误则应该中止导出。

当然，在使用 EXP 程序时也可以不使用任何参数，此时需要用户输入一些参数来导出要备份的数据，如用户名和密码、输入缓冲区大小、导出的文件名、导出表或者导出所有用户对象、是否导出权限等。在以下几节中我们介绍如何使用 EXP 导出整个数据库、导出特定的用户、导出特定的表和特定的表空间。

9.2.2　不带参数的 EXP 备份

在上节的最后部分，我们提到了可以使用不带参数的 EXP 指令备份数据库，不过系统会提示输入一系列参数完成备份。本节给出一个例子以更详细的理解这些参数，学会使用这种方式备份数据库，如下例所示。

例子 9-2 使用 EXP 实用程序备份 SCOTT 用户的所有数据库对象。

```
C:\Documents and Settings\Administrator>exp

Export: Release 10.1.0.6.0 - Production on 星期五 10月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

用户名: scott@orcl
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
输入数组提取缓冲区大小: 4096 >

 导出文件: EXPDAT.DMP >

(2)U(用户), 或 (3)T(表): (2)U >

导出权限 (yes/no): yes >

导出表数据 (yes/no): yes >

压缩区 (yes/no): yes >
```

```
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集
. 正在导出 pre-schema 过程对象和操作
. 正在导出用户 SCOTT 的外部函数库名
. 导出 PUBLIC 类型同义词
. 正在导出专用类型同义词
. 正在导出用户 SCOTT 的对象类型定义
即将导出 SCOTT 的对象...
. 正在导出数据库链接
. 正在导出序号
. 正在导出簇定义
. 即将导出 SCOTT 的表通过常规路径...
. . 正在导出表        BACKUP DELETE EMP TABLE 导出了          0 行
. . 正在导出表                          BONUS 导出了          0 行
. . 正在导出表          CREATE$JAVA$LOB$TABLE 导出了          1 行
. . 正在导出表                           DEPT 导出了          4 行
. . 正在导出表                      DEPT TEST 导出了          5 行
. . 正在导出表                            EMP 导出了         28 行
. . 正在导出表                      EMP TEST2 导出了          0 行
. . 正在导出表                      EMP TEST3 导出了         2 行
. . 正在导出表                          ROOMS 导出了        4 行
. . 正在导出表                       SALGRADE 导出了         5 行
. . 正在导出表             USER MODIFY TABLE 导出了         20 行
. 正在导出同义词
. 正在导出视图
. 正在导出存储过程
. 正在导出运算符
. 正在导出引用完整性约束条件
. 正在导出触发器
. 正在导出索引类型
. 正在导出位图，功能性索引和可扩展索引
. 正在导出后期表活动
. 正在导出实体化视图
. 正在导出快照日志
. 正在导出作业队列
. 正在导出刷新组和子组
. 正在导出维
. 正在导出 post-schema 过程对象和操作
. 正在导出统计信息
成功终止导出，没有出现警告。
```

EXP 指令是在操作系统下运行的，导出的文件保存在运行 EXP 指令的当前操作系统目录下。一旦输入 EXP 指令然后回车，就提示输入用户名和密码，如果二者都正确，则提示连接到数据库，然后是一系列的参数选择。下面依次解释这些参数。

- 输入数组提取缓冲区大小: 4096 >
  缓冲区大小在 WINDOWS 上默认是 4096 字节，如果用户包含大对象(LOB)，则设置该参数至少为 2M。
- 导出文件: EXPDAT.DMP >
  导出的备份文件默认文件名为 EXPDAT.DMP，也可以根据需要自己命名，在>后输入自定义的导出备份文件名。

- (2)U(用户), 或 (3)T(表): (2)U >
  导出当前用户的所有数据库对象，还是导出当前用户的表，冒号后是默认值，默认是导出用户的数据库对象。如果选择导出表，接下来则提示如下：

```
(2)U(用户), 或 (3)T(表): (2)U > t

导出表数据 (yes/no): yes >

压缩区 (yes/no): yes >

已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的表通过常规路径...
要导出的表 (T) 或分区 (T: P): (按 RETURN 退出) > dept

. . 正在导出表                          DEPT 导出了          4 行
```

  使用这种方式导出用户表时，每次只能导出当前用户的一个表，所以如果有多个表要导出则需要多次完成这些操作步骤。读者根据自己的需求选择是否使用这种方式。
- 导出权限(yes/no): yes >: 选择是否导出权限，即是否导出对表、视图、序列或角色的授权。
- 导出表数据 (yes/no): yes >
  是否导出表数据，Oracle 提供了这样一种方式既可以导出表的结构也可以导出数据，或者导出表结构和表数据。如果不导出表数据就是只导出表的结构，也就是导出表的定义，模式选择导出表数据。
- 压缩区 (yes/no): yes >: 选择是否使用压缩区。

9.2.3 EXP 指令导出整个数据库

导出整个数据库比较简单，只需要使用 FULL 和 FILE 两个参数。FULL 表示导出整个文件，包括所有的数据库对象，而 FILE 表示备份后的数据库文件名，可以自定义。使用 SYSTEM 用户登录数据库，导出整个数据库，如下例所示。

例子 9-3 EXP 指令备份整个数据库。

```
D:\>exp system/oracle@orcl full=y file = backup_wholedatabase.dmp

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0  - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出整个数据库...
. 正在导出表空间定义
. 正在导出概要文件
. 正在导出用户定义
```

```
. 正在导出角色
. 正在导出资源成本
. 正在导出回退段定义
. 正在导出数据库链接
. 正在导出序号
. 正在导出目录别名
. 正在导出上下文名称空间
. 正在导出外部函数库名
. 导出 PUBLIC 类型同义词
. 正在导出专用类型同义词
. 正在导出对象类型定义
. 正在导出系统过程对象和操作
. 正在导出 pre-schema 过程对象和操作
. 正在导出簇定义
. 即将导出 SYSTEM 的表通过常规路径...
. . 正在导出表                    DEF$ AQCALL 导出了          0 行
. . 正在导出表                   DEF$ AQERROR 导出了          0 行
。。。。。。。。。
. . 正在导出分区                     COSTS 1995 导出了          0 行
. . 正在导出分区                     COSTS 1996 导出了          0 行
。。。。。。。。。
. 即将导出 PM 的表通过常规路径...
. . 正在导出表                   ONLINE MEDIA 导出了          9 行
. . 正在导出表                    PRINT MEDIA 导出了          4 行
. . 正在导出表              TEXTDOCS NESTEDTAB 导出了         12 行
. 即将导出 BI 的表通过常规路径...
. 即将导出 CAT 的表通过常规路径...
. 正在导出同义词
. 正在导出视图
. 正在导出引用完整性约束条件
. 正在导出存储过程
. 正在导出运算符
. 正在导出索引类型
. 正在导出位图，功能性索引和可扩展索引
. 正在导出后期表活动
. 正在导出触发器
. 正在导出实体化视图
. 正在导出快照日志
. 正在导出作业队列
. 正在导出刷新组和子组
. 正在导出维
. 正在导出 post-schema 过程对象和操作
. 正在导出用户历史记录表
. 正在导出默认值和系统审计选项
. 正在导出统计信息
导出成功终止，但出现警告。
```

输出显示导出成功终止，备份的全库文件名为 backup_wholedatabase_090810.dmp，从备份输出信息可以看出，开始导出了数据库对象的定义如表空间定义、概要文件定义以及用户定义，然后导出数据库中的表、分区，最后导出数据库中各类对象的定义。总之，通过全库导出，导出了数据库中对象的定义，以及各种表结构以及表中的数据，这种备份也可以称为逻辑备份。

如果只需要导出数据库的结构而不需要表中的数据，则可以使用如下例所示的指令。

例子 9-4 使用 EXP 指令导出不包含数据的全库备份。

```
D:\>exp system/oracle@orcl full =y rows =n file =myfull_withoutdata.dmp

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Produ
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集
注: 将不导出表数据 (行)

即将导出整个数据库...
. 正在导出表空间定义
......
```

这里我们为了节约篇幅，只给出部分输出结果，省略的部分和例 9-7 中的相应部分相同，唯一的区别是只导出对象定义，而不导出数据。

参数 ROWS=no，告诉 EXP 程序要导出的数据库不包含数据，只导出整个数据库中数据库对象的定义。

9.2.4 EXP 指令导出特定的表

导出特定用户的特定的表需要 TABLES 参数，该参数后可以有几个表名，如果需要导出的表不是当前用户的表，则需要使用 schema_name.table_name 的形式，告诉 EXP 程序这表属于哪个用户。使用 FILE 参数自定义备份文件名。下面给出一个例子，使用 SYSTEM 用户登录，导出 SCOTT 用户的两个表 DEPT 和 EMP。

例子 9-5 使用 EXP 指令导出特定的表。

```
D:\>exp system/oracle@orcl tables = scott.dept,scott.emp file = tables.dmp

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的表通过常规路径...
当前的用户已更改为 SCOTT
. . 正在导出表                          DEPT 导出了          4 行
. . 正在导出表                           EMP 导出了         28 行
成功终止导出, 没有出现警告。
```

在输出中显示导出成功，导出的文件名为 tables.dmp。因为用户 SYSTEM 具有对用户 SCOTT 中表的访问权，所以该用户可以导出 SCOTT 用户的表，如果使用 SCOTT 用户登录数据库，在导出自己的表时可以直接写出表名，而不必使用模式名。如下例所示，导出 SCOTT 用户的两个表 DEPT 和 EMP。

例子 9-6 使用 EXP 导出当前用户的表。

```
D:\>exp scott/oracle@orcl tables = dept,emp file = backup_scott_tables.dmp

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的表通过常规路径...
. . 正在导出表                          DEPT 导出了          4 行
. . 正在导出表                           EMP 导出了         28 行
成功终止导出，没有出现警告。
```

上例中，使用 SCOTT 用户登录，而导出的表也是该用户拥有的表，所以不需要指定模式名。

9.2.5 EXP 指令导出指定的用户

如果不需要导出整个数据库，而是导出当前数据库中某个用户的所有数据库对象，作为该用户的数据库备份，则使用 OWNER 参数指定，如下例所示。

例子 9-7 使用 EXP 指令导出 SCOTT 用户的所有数据库对象。

```
D:\>exp system/oracle@orcl owner = scott file = userbackup scott.dmp

Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的用户...
. 正在导出 pre-schema 过程对象和操作
. 正在导出用户 SCOTT 的外部函数库名
. 导出 PUBLIC 类型同义词
. 正在导出专用类型同义词
. 正在导出用户 SCOTT 的对象类型定义
即将导出 SCOTT 的对象...
. 正在导出数据库链接
```

```
. 正在导出序号
. 正在导出簇定义
. 即将导出 SCOTT 的表通过常规路径...
. . 正在导出表        BACKUP_DELETE_EMP_TABLE 导出了       0 行
. . 正在导出表                          BONUS 导出了       0 行
. . 正在导出表         CREATE$JAVA$LOB$TABLE 导出了       1 行
. . 正在导出表                           DEPT 导出了       4 行
. . 正在导出表                      DEPT_TEST 导出了       5 行
. . 正在导出表                            EMP 导出了       28 行
. . 正在导出表                      EMP_TEST2 导出了       0 行
. . 正在导出表                      EMP_TEST3 导出了       2 行
. . 正在导出表                          ROOMS 导出了       4 行
. . 正在导出表                       SALGRADE 导出了       5 行
. . 正在导出表              USER_MODIFY_TABLE 导出了       20 行
. 正在导出同义词
. 正在导出视图
. 正在导出存储过程
. 正在导出运算符
. 正在导出引用完整性约束条件
. 正在导出触发器
. 正在导出索引类型
. 正在导出位图，功能性索引和可扩展索引
. 正在导出后期表活动
. 正在导出实体化视图
. 正在导出快照日志
. 正在导出作业队列
. 正在导出刷新组和子组
. 正在导出维
. 正在导出 post-schema 过程对象和操作
. 正在导出统计信息
成功终止导出，没有出现警告。
```

导出特定用户的所有数据库对象，其实是导出该用户所拥有的数据库对象的定义，以及对象所拥有的数据。从导出输出中可以清楚的看到，已经导出了这些对象的定义，以及表中的数据。

上例中我们使用 SYSTEM 用户登录数据库，该用户具有 DBA 权限，所以它拥有访问 SCOTT 用户的数据库对象权限，可以成功导出 SCOTT 用户的所有数据库对象。其实，也完全可以使用 SCOTT 用户登录数据库，使用 EXP 指令导出它自己拥有的所有数据库对象，如下例所示。

例子 9-8 使用 EXP 指令导出当前用户的数据库对象。

```
D:\>exp scott/oracle@orcl owner = scott file = bakup_user_scott.dmp
```

此处，我们只给出指令，而不显示具体结果，其备份过程与例 9-7 的输出一样。

9.2.6 EXP 指令导出特定的表空间

在数据库维护时，可能只需要重点维护一个表空间，如 USERS 表空间，该表空间放置用户数据。此时可使用 EXP 指令的 TABLESPACES 参数来指定当前用户所拥有的特定表空间，如下例所示。

例子 9-9 使用 EXP 指令导出特定表空间。

```
    D:\>exp system/oracle@orcl tablespaces =users file = backup_tablespaces_
users.dmp

    Export: Release 10.1.0.6.0 - Production on 星期五 10 月 16 15:40:20 2009

    Copyright (c) 1982, 2007, Oracle. All rights reserved.


    连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

    即将导出所选表空间...
    对于表空间 USERS...
    . 正在导出簇定义
    . 正在导出表定义
    . . 正在导出表       BACKUP DELETE EMP TABLE 导出了          0 行
    . . 正在导出表                        BONUS 导出了           0 行
    . . 正在导出表         CREATE$JAVA$LOB$TABLE 导出了          1 行
    . . 正在导出表                         DEPT 导出了           4 行
    . . 正在导出表                    DEPT TEST 导出了           5 行
    . . 正在导出表                          EMP 导出了          28 行
    . . 正在导出表                    EMP TEST2 导出了           0 行
    . . 正在导出表                    EMP TEST3 导出了           2 行
    . . 正在导出表                        ROOMS 导出了           4 行
    . . 正在导出表                     SALGRADE 导出了           5 行
    . . 正在导出表           USER MODIFY TABLE 导出了           20 行
    . . 正在导出表               CATEGORIES TAB 导出了          22 行
    . . 正在导出表     PRODUCT REF LIST NESTEDTAB 导出了        288 行
    . . 正在导出表 SUBCATEGORY REF LIST NESTEDTAB 导出了        21 行
    EXP-00079: 表 "PURCHASEORDER" 中的数据是被保护的。常规路径只能导出部分表。
    . . 正在导出表                PURCHASEORDER 导出了         132 行
    . 正在导出引用完整性约束条件
    . 正在导出触发器
    导出成功终止, 但出现警告。
```

EXP 将针对特定的表空间实施导出操作，导出该表空间中所有表的定义和表中的数据。如果不需要导出表中的数据，只需要备份表空间中对象的定义，可以使用 ROWS 参数，设置 rows = no 即可。

9.3　使用IMP指令实现逻辑恢复

IMP 指令能引导用户导入通过 EXP 指令导出的备份数据，IMP 可以导入一个完整的数据库、一个指定的用户的所有数据库对象、一个特定的表空间以及一个特定的表。在说明如何使用 IMP 指令导入数据前，我们先说明 IMP 指令的参数指令，再使用 IMP 指令实现数据库的恢复。

9.3.1　IMP 指令详解

IMP 指令同样是运行在操作系统上的，通过输入 IMP 指令和各种参数来控制数据导入的运行方式。在 WINDOWS 的 DOS 窗口中输入 emp help =y ，然后回车，会显示如下例所示的 IMP 参数信息。

例子 9-10 IMP 实用程序的参数指令。

```
C:\Documents and Settings\Administrator>imp help=y

Import: Release 10.1.0.6.0 - Production on 星期二 8月 10 07:55:12 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

通过输入 IMP 命令和您的用户名/口令, 导入
操作将提示您输入参数:

    例如: IMP SCOTT/TIGER

或者, 可以通过输入 IMP 命令和各种参数来控制导入
的运行方式。要指定参数, 您可以使用关键字:

    格式:  IMP KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)
    例如: IMP SCOTT/TIGER IGNORE=Y TABLES=(EMP,DEPT) FULL=N
            或 TABLES=(T1:P1,T1:P2), 如果 T1 是分区表

USERID 必须是命令行中的第一个参数。

关键字        说明 (默认值)          关键字            说明 (默认值)
--------------------------------------------------------------------------
USERID       用户名/口令            FULL             导入整个文件 (N)
BUFFER       数据缓冲区大小          FROMUSER         所有者用户名列表
FILE         输入文件 (EXPDAT.DMP)  TOUSER           用户名列表
SHOW         只列出文件内容 (N)      TABLES           表名列表
IGNORE       忽略创建错误 (N)        RECORDLENGTH     IO 记录的长度
GRANTS       导入权限 (Y)           INCTYPE          增量导入类型
INDEXES      导入索引 (Y)           COMMIT           提交数组插入 (N)
ROWS         导入数据行 (Y)          PARFILE          参数文件名
LOG          屏幕输出的日志文件       CONSTRAINTS      导入限制 (Y)
DESTROY                    覆盖表空间数据文件 (N)
INDEXFILE                  将表/索引信息写入指定的文件
SKIP UNUSABLE INDEXES      跳过不可用索引的维护 (N)
FEEDBACK                   每 x 行显示进度 (0)
TOID NOVALIDATE            跳过指定类型 ID 的验证
FILESIZE                   每个转储文件的最大大小
STATISTICS                 始终导入预计算的统计信息
RESUMABLE                  在遇到有关空间的错误时挂起 (N)
RESUMABLE NAME             用来标识可恢复语句的文本字符串
RESUMABLE TIMEOUT          RESUMABLE 的等待时间
COMPILE                    编译过程, 程序包和函数 (Y)
STREAMS_CONFIGURATION      导入流的一般元数据 (Y)
```

```
STREAMS_INSTANTIATION   导入流实例化元数据 (N)

下列关键字仅用于可传输的表空间
TRANSPORT_TABLESPACE 导入可传输的表空间元数据 (N)
TABLESPACES 将要传输到数据库的表空间
DATAFILES 将要传输到数据库的数据文件
TTS_OWNERS 拥有可传输表空间集中数据的用户

成功终止导入，没有出现警告。
```

下面详细解释上述输出中部分参数的含义：

- USERID：说明登录数据库的用户名和密码。
- BUFFER：说明输入数据缓冲区的大小。
- FILE：说明通过 EXP 程序创建的备份文件名，有时需要绝对路径。
- SHOW：说明是否使得导入过程只显示备份文件的内容。
- IGNORE：说明是否忽略在输入过程中备份文件中的错误。
- GRANTS：说明是否导入备份文件中的授权。
- INDEXES：说明是否导入索引。
- ROWS：是否导入数据行，该参数的值为 y 或者 n，如果 ROWS = n，则不输入数据。
- LOG：是否将导入过程记录到日志文件。
- FULL：对整个备份文件的完全导入。
- FROMUSER：允许将一个备份文件中的对象从一个用户复制到另一个用户，该参数说明导入数据的源用户名。
- TOUSER：允许将一个备份文件中的对象从一个用户复制到另一个用户，该参数说明导入数据的目的户名。
- TABLES：说明要导入的表名列表，如果是多个表使用逗号分开。
- SKIP_UNUSABLE_INDEXES：该参数值为 y 或 n，说明是否需要重建已经设置为 unusable 状态的索引。
- STATISTICS：说明在导入对象后对统计量的如何处理，该参数值为 always、none、safe 或者 recalculate。其中 always 表示从备份文件中导入统计量；none 表示不执行任何动作；safe 表示如果使用了可靠的优化器，则允许导入优化器统计量；recalculate 表示需要重新计算统计量而不是从备份文件中导入统计量。
- CONSTRAINTS：说明是否导入备份数据中的约束。
- COMPILE：该参数说明是否引入进程来编译过程、包和函数。

9.3.2　IMP 指令恢复整个数据库

IMP 使用程序将备份的整个数据库导入当前的数据库中。在导入过程中包括一系列创建数据库对象的过程，如创建表空间、表、表的索引、序列号以及用户数授权等。如果要创建的数据库对象如某个表已经存在，则该创建语句会失败。此时需要使用 ignore 参数，该参数使得 IMP 程序忽略这些错误。下面是使用了 ignore 参数进行全库恢复的例子。

例子 9-11 使用 IMP 指令导入整个数据库。

```
D:\>imp system/oracle@orcl full =y file=backup_wholedatabase.dmp ignore =y
```

上例中参数 file 后的文件名说明该文件位于当前使用 IMP 指令的目录下，参数 ignore=y 使得在创建数据库对象时，不会造成由于对象已经存在导致输入操作产生错误。

如果用户的数据量比较大，这个过程将很“漫长”，使用这种方法恢复整个数据库其实就是重新创建数据库中所有对象的过程，所以它是一种逻辑数据库恢复的方法。

9.3.3 IMP 指令恢复特定的表

在 9.2.4 节中，我们使用 EXP 指令导出了用户 SCOTT 的两个表 DEPT 和 EMP。如果由于某种原因删除了表 EMP，然后需要恢复 EMP 表（当然可以通过 Oracle 的脚本文件重建 SCOTT 用户的所有数据库对象，这里模拟恢复特定的表），那么可以使用 EXP 程序备份的导出文件 backup_scott_tables.dmp。我们先删除 SCOTT 用户的两个表，如下例所示。

例子 9-12 删除用户 SCOTT 的 EMP 表。

```
SQL> drop table emp;

表已删除。
```

然后查询该表是否存在，如下例所示。

例子 9-13 验证表 EMP 是否存在。

```
SQL> desc emp;
ERROR:
ORA-04043: 对象 emp 不存在
```

显然，输出结果说明我们已经成功删除了表 EMP，下面我们使用 IMP 程序来恢复找个表。如下例所示，使用 IMP 指令恢复特定的表。

例子 9-14 使用 IMP 指令恢复特定的表。

```
D:\>imp scott/oracle@orcl tables =emp file =backup_scott_tables.dmp

Import: Release 10.1.0.6.0 - Production on 星期二 8 月 10 23:27:21 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.1.0.6.0 创建的导出文件
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
. 正在将 SCOTT 的对象导入到 SCOTT
. 正在将 SCOTT 的对象导入到 SCOTT
. . 正在导入表                  "EMP"导入了          28 行
成功终止导入，没有出现警告。
```

显然，成功从备份文件 backup_scott_tables.dmp 中恢复了用户 SCOTT 的表 EMP。

下面再通过 DESC 指令查看表 EMP 的定义是否存在，如下例所示。

例子 9-15 查看恢复后表 EMP 是否存在。

```
SQL> desc emp;
 名称                                        是否为空? 类型
 ----------------------------------------- -------- --------------
 EMPNO                                              NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

9.3.4　IMP 指令恢复指定的用户

在一个数据库中可以创建多个用户，每个用户有自己的数据库对象，如表、表空间、表索引、序列号、约束等。使用 EXP 程序导出指定的用户，其实就是导出该用户所拥有的所有数据库对象。如果某用户的数据库对象需要恢复，则可是使用 IMP 指令恢复用户的整个数据库对象、部分表，或者将该用户的表导入其他用户。导入 SCOTT 用户的全部数据库对象，如下例所示。

例子 9-16 导入 SCOTT 用户的全部数据库对象。

```
D:\>imp system/oracle@orcl full =y file =userbackup_scott.dmp

Import: Release 10.1.0.6.0 - Production on 星期三 8 月 26 00:04:41 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release 10.1.0.6.0 - Productio
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.1.0.6.0 创建的导出文件
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
. 正在将 SYSTEM 的对象导入到 SYSTEM
. 正在将 SCOTT 的对象导入到 SCOTT
. . 正在导入表      "BACKUP_DELETE_EMP_TABLE"导入了          0 行
. . 正在导入表                      "BONUS"导入了          0 行
. . 正在导入表        "CREATE$JAVA$LOB$TABLE"导入了          1 行
. . 正在导入表                       "DEPT"导入了          4 行
. . 正在导入表                  "DEPT_TEST"导入了          5 行
. . 正在导入表                        "EMP"导入了         28 行
. . 正在导入表                  "EMP_TEST2"导入了          0 行
. . 正在导入表                  "EMP_TEST3"导入了          2 行
. . 正在导入表                      "ROOMS"导入了          4 行
. . 正在导入表                   "SALGRADE"导入了          5 行
. . 正在导入表          "USER_MODIFY_TABLE"导入了         20 行
```

```
即将启用约束条件...
成功终止导入，没有出现警告。
```

在上例中，我们使用了 9.2.5 节中导出的用户 SCOTT 的用户对象数据，导入整个数据库对象。因为该文件当时是具有 DBA 权限的用户导出的，所以导入时也必须使用 DBA 权限的用户导入。

此例中必须指定 FULL 参数。如果需要导入特定的表数据，可以使用 TABLES 参数，这样可以把特定的表导入到新的用户，如下例所示。

例子 9-17 将特定的表导入指定用户。

```
D:\>imp system/oracle@orcl tables =emp fromuser=scott touser=system file
=userbackup_scott.dmp

Import: Release 10.1.0.6.0 - Production on 星期三 8 月 26 00:20:46 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


连接到: Oracle Database 10g Enterprise Edition Release10.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.1.0.6.0 创建的导出文件
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
. 正在将 SCOTT 的对象导入到 SYSTEM
. . 正在导入表                      "EMP"导入了        28 行
"CREATE TRIGGER "SYSTEM".user_change_empdata"
 "before update or insert on emp"
 "for each row"
 "begin"
 " if inserting then"
 "insert into user_modify_table"
 " values (user,sysdate,'inserting');"
 "end if;"
 "if updating then"
 "insert into user_modify_table"
 " values (user,sysdate,'updating');"
 "end if;"
 "end user change empdata;"
 "CREATE TRIGGER "SYSTEM".when_backup_emp_trigger"
 "before delete on emp"
 "for each row"
 "when (old.job IN 'MANAGER,PRESIDENT')"
 "begin"
 "insert into backup delete emp table"
 "values(:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,"
 "      :old.sal,:old.comm,:old.deptno);"
 "end when backup emp trigger;"
成功终止导入，没有出现警告。
```

在上例中，我们向用户 SYSTEM 导入了 SCOTT 用户的表 EMP。当导入 EMP 表时，其实就是在新用户 SYSTEM 的用户表空间中创建一个表，并填充数据，最后创建基于该表的触发器。如果读者自己创建了一个用户，一定要注意该用户必须具有对存放新表的表空间的访问权，否则提示

对表空间无权限，无法恢复数据库对象到指定的用户。

9.4　使用EXP/IMP实现传输表空间

从 Oracle 10g 开始支持跨平台的表空间传输，可以想象既然 Oracle 是一款跨平台的数据库软件，开始支持跨平台间的表空间传输也是情理之中。表空间传输是实现数据迁移的一种方法。

传统方式中，如果一个表空间中的所有数据需要迁移到另一个平台，往往需要使用 EXP/IMP 数据库导入导出工具迁移，这种方式花费较多的恢复时间，而使用传输表空间则可以很快速地恢复表空间。

使用传输表空间实现在两个数据库之间拷贝表空间数据集（表空间逻辑结构和数据），传输表空间可以使用数据字典管理或者本地管理。从 Oracle 9i 开始，传输表空间不再要求传输表空间的数据块大小与目标数据库的标准数据块大小相同。使用传输表空间移动数据比使用导入/导出工具集或者其他装载工具如 SQL*LOADER 要快得多，因为数据文件包含实际的数据，而数据文件使用 OS 工具直接传输到目标数据库，只需要使用导入/导出工具传输表空间对象的元数据（如表的定义，索引等）到目标数据库。其中，导入/导出工具可以是数据泵也可以是 IMP/EXP 工具。

在以下场合多使用传输表空间。

- 导出或导入数据仓库表分区。
- 发布存储在 CD 上的结构化数据。
- 在多个数据库之间拷贝同一个表空间的多个只读版本。
- 归档历史数据。
- 执行表空间的时间点恢复。

实现传输表空间可以使用手工方法，即使用 SQL*PLUS 工具，RMAN 以及 EXP/IMP 工具或数据泵工具。也可以使用 OEM 在企业管理器中使用传输表空间 WIZARD，运行传输表空间 WIZARD。本节说明如何通过 SQL*PLUS 工具实现传输表空间。

9.4.1　理解 Big/Little Endian

在传输表空间时需要注意“字节存储序列”，因为在不同的平台上它是不同的。字节序列决定了数据值在操作系统平台上的存储顺序，有 Big Endian 和 Little Endian 之分。

其中 Big Endian 是大值存储位靠前，而 Little Endian 得大值存储位靠后。如果 1234 在支持 Big Endian 的操作系统中存储顺序是 3412，而在支持 Little Endian 的操作系统中存储顺序是 1234。

在不同平台传输表空间之前，要清楚两个平台的直接序列是否相同，如果不同还需要字节序列转换来转换数据文件的存储序列，该内容我们会在说明跨平台表空间迁移时详细说明。

9.4.2　传输表空间的限制

在上节我们介绍了传输表空间及其带来的好处，但是不能随意使用传输表空间，使用传输表空间需要两个数据库平台一些特定要求，这些限制说明如下。

- 源和目标数据库必须使用相同的字符集和国家字符集。
- 如果目标数据库有一个与传输表空间相同名称的表空间存在，则不能传输该表空间到目标数据库。所以，在这种情况下，你需要在源数据库端或目标数据库端修改表空间名（RENAME），之后再进行表空间的传输。
- 如果一些对象具有更底层的对象如物化视图，或带有包含对象如分区表，这样的对象不能被传输，除非所有的底层对象或者包含对象在传输表空间集中。
- 从 Oracle 10g 开始，可以传输具有 XMLTypes 的传输表空间，但是必须使用 IMP/EXP 工具，而不是使用数据泵。当使用 EXP 工具时，确保 CONSTRAINTS 和 TRIGGER 参数设置为 Y（也是默认值）。

可以执行如下的查询确认包含 XMLTypes 的表空间。

```
SQL> select distinct p.tablespace name from dba tablespaces p,
  2  dba_xml_tables x, dba_users u, all_all_tables t where
  3  t.table name=x.table name and t.tablespace name=p.tablespace name
  4  and x.owner=u.username;

TABLESPACE NAME
------------------------------
SYSAUX
```

从输出知道，目前只用 SYSAUX 辅助表空间具有 XMLTypes。下面我们详细分析具有 XMLTypes 类型的传输表空间限制。

- 目标数据库必须安装了 XML DB。
- 应用 XMLTypes 类型表的模式不能是 XML DB 的标准模式。
- 应用 XMLTypes 类型表的模式不能有循环依赖。
- 对于 IMPORT 指令，任何 XMLTypes 类型表的行级安全都将丢失，因为实现行级安全的访问列表不能被导入，这是由于目标数据库不一定具有和源数据库相同的用户。
- 如果带有 XMLTypes 的表的模式在目标数据库中不存在，则导入并注册新用户，如果已经存在，则可以使用 IGNORE=Y 来忽略存在用户。
- 不能传输 SYSTEM 表空间以及 SYS 用户拥有的数据库对象，这些数据库对象如 PL/SQL、JAVA 类、视图、同义词、用户、权限、目录以及序列号等。

9.4.3　传输表空间的兼容性问题

一旦在目标数据库端创建了传输表空间集，Oracle 数据库计算目标数据库需要运行的最低数据库版本，即传输表空间的最小兼容性。Oracle 数据库的表空间总可以传输到相同或更高版本的数据库，而不需要考虑目标数据库运行的操作系统平台。

9.4.4　传输表空间的自包含特性

在一个传输表空间中，不同的数据库对象之间可能会有逻辑或物理的依赖关系。传输表空间中对象的某些依赖对象并不存在于传输表空间中，这样的表空间不能传输。Oracle 只允许传输自包

含的表空间，自包含的意思是传输表空间中没有引用存在于传输表空间之外的对象。以下几种情况违反了表空间的自包含特性。

- 传输表空间中存储了另一个表空间中表的索引，这个索引对象不违反自包含特性。但是，如果传输表空间中存在一个表，而该表的索引存在于另一个表空间中，则该传输表空间就是自包含的。
- 分区表部分在传输表空间中，这种情况也是违反表空间的自包含，必须将分区表的所有分区都存储在传输表空间中，或者将分区表转换成实体表。
- 传输表空间中的表包含 LOG 列，该列指向传输表空间外的 LOG 对象。

Oracle 允许使用 DBMS_TTS 包的 TRANSPORT_SET_CHECK 过程验证表空间是否具有自包含特性，执行该过程需要用户具有 EXECUTE_CATALOG_ROLE 角色，该角色最初赋予了 SYS 用户。

9.4.5　实现传输表空间的步骤

要实现传输表空间必须遵循 Oracle 的步骤进行，下面是实现传输表空间的具体步骤。读者必须严格按照这个步骤进行，验证传输表空间是否满足要求的条件。

01 对于跨平台的表空间传输需要使用 V$TRANSPORTABLE_PLATFORM 查询每个平台支持的 ENDIANESS，如下例所示。

例子 9-18　查询当前平台的 ENDIANESS。

```
SQL> col platform name for a30
SQL> select *
  2* from v$transportable platform

PLATFORM_ID PLATFORM_NAME                           ENDIAN_FORMAT
--------------------- ------------------------------ --------------
        1 Solaris[tm] OE (32-bit)                   Big
        2 Solaris[tm] OE (64-bit)                   Big
        7 Microsoft Windows IA (32-bit)             Little
       10 Linux IA (32-bit)                         Little
        6 AIX-Based Systems (64-bit)                Big
        3 HP-UX (64-bit)                            Big
        5 HP Tru64 UNIX                             Little
        4 HP-UX IA (64-bit)                         Big
       11 Linux IA (64-bit)                         Little
       15 HP Open VMS                               Little
        8 Microsoft Windows IA (64-bit)             Little

PLATFORM_ID PLATFORM_NAME                           ENDIAN_FORMAT
----------- ------------------------------ --------------
        9 IBM zSeries Based Linux                    Big
       13 Linux 64-bit for AMD                       Little
       16 Apple Mac OS                               Big
       12 Microsoft Windows 64-bit for A Little
          MD
```

```
        17 Solaris Operating System (x86) Little
        18 IBM Power Based Linux                    Big

已选择 17 行。
```

这个步骤，说明当前的数据库版本支持的不同操作系统平台，也就是告诉我们可以在这些操作系统平台之间进行表空间的迁移。

02 查询当前的传输表空间是否是自包含的，如下例所示。

例子 9-19 查询传输表空间的自包含。

```
SQL> execute dbms_tts.transport_set_check('test2',true);

PL/SQL 过程已成功完成。

SQL> select *
  2  from transport_set_violations;

VIOLATIONS
---------------------------------------------------------------------------
Sys owned object  TEST2_EMP in tablespace TEST2 not allowed in pluggable set
```

因为 TEST2_EMP 是 SYS 拥有的表对象，所以不允许迁移。我们删除表空间 TEST 中的表对象。使用 SCOTT 用户创建新表，如下例所示。

例子 9-20 创建用户 SCOTT 的新表。

```
SQL> create table test2 emp tablespace test2
  2  as
  3  select *
  4  from emp;

表已创建。
```

切换到 SYS 用户，继续执行自包含检查，并查看 transport_set_violations 视图以确定是否有违反自包含特性的对象，如下面例子所示。

例子 9-21 执行表空间自包含检查。

```
SQL> conn sys/oracle as sysdba
已连接。
SQL> execute dbms_tts.transport_set_check('test2',true);

PL/SQL 过程已成功完成。
```

例子 9-22 查看是否有违反自包含特性的对象。

```
SQL> select *
  2  from transport_set_violations;

未选定行
```

此时，发现没有任何违反表空间自包含特性的输出。

03 使用 EXP 或者 EXPDP 等工具导出表空间元数据。

元数据是表空间的定义和表的结构，它不包含实际的表数据。使用 EXP 工具将表空间的元数据集合导出，此时需要先将表空间 READ ONLY。

如果此时的目标数据库与源数据的字节序列（NEDIANESS）不同，则需要将导出的表空间元数据集转换为目的数据库的 ENDIANESS。这个转换操作既可以在源数据库端执行，也可以在目的数据库端执行。

04 备份该表空间的数据文件。可以使用操作系统工具备份传输表空间的数据文件。

05 将传输表空间的导出元数据集和数据文件拷贝到目的数据库。

06 使用 EXP 或者 IMPDP 工具将传输表空间的元数据导入目的数据库。

9.4.6　使用 EXP/IMP 实现同平台表空间迁移

同平台之间的转换很简单，因为不必考虑 ENDIANESS 的问题，读者也熟悉同平台的数据迁移方法，比如 CP 指令或者 FTP 指令的使用相同操作系统平台之间表空间的迁移流程示意图，如图 9-1 所示。

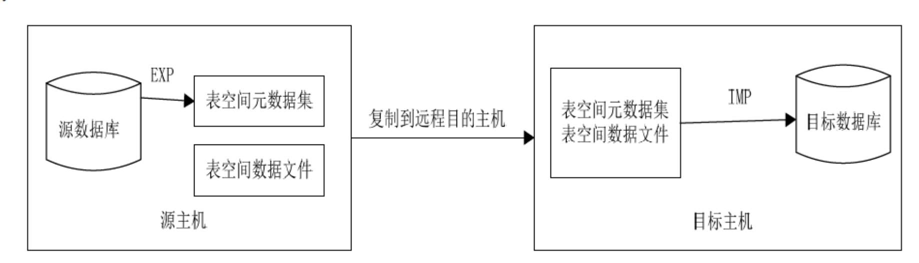


图 9-1　同 OS 平台表空间迁移流程。

在相同数据库平台上进行表空间的迁移，不需要考虑操作系统的 ENDIANESS，并且我们假设两个数据库平台的版本相同，或者目标数据库平台版本比源数据库平台版本要高。如果目标数据库版本为 10g，而源数据库版本为 9i，则这样的迁移是兼容的。我们做测试时，使用相同数据库版本的迁移，二者的数据库平台都为 10g。

我们解释一下图 9-1 的流程。在迁移时，源主机上使用 EXP 工具将迁移表空间的源数据集备份下来，然后将该备份集与表空间对应的数据文件的二进制文件统一拷贝到目标主机。在目标主机端使用 IMP 工具将表空间的源数据集和数据文件导入目标数据库。

下面通过一个例子演示这个过程。两个数据库都是安装在 WINDOWS 平台，两个数据库分别为 ORCL1 和 ORCL2，在数据库 ORCL2 上创建传输表空间 TEST2，在其中存放了表 TEST2_EMP，该表的拥有者为 SCOTT。

01 表空间自包含检查，如下例所示。

例子 9-23 表空间 TEST2 的自包含检查。

```
SQL> conn sys/oracle as sysdba
已连接。
SQL> execute dbms_tts.transport_set_check('test2',true);

PL/SQL 过程已成功完成。
```

```
SQL> select *
  2  from transport_set_violations;

未选定行
```

如果在数据字典 transport_set_violations 记录了违反自包含的数据库对象，则修改这些问题。然后继续下一步。

02 将表空间设置为 READ ONLY 并备份表空间的元数据集，如下面例子所示。

例子 9-24 将表空间设置为 READ ONLY。

```
SQL> alter tablespace test2 read only;

表空间已更改。
```

例子 9-25 导出表空间 test2 的元数据。

```
D:\>exp sys/oracle file=expdp.dmp tablespaces=test2 transport_tablespace=y

Export: Release 10.2.0.1.0 - Production on 星期五 7月 2 11:34:46 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


EXP-00056: 遇到 ORACLE 错误 28009
ORA-28009: connection as SYS should be as SYSDBA or SYSOPER
用户名: sys as sysdba
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集
注: 将不导出表数据 (行)
即将导出可传输的表空间元数据...
对于表空间 TEST2...
. 正在导出簇定义
. 正在导出表定义
. . 正在导出表                    TEST2 EMP
. 正在导出引用完整性约束条件
. 正在导出触发器
. 结束导出可传输的表空间元数据
成功终止导出, 没有出现警告。
```

03 将表空间的元数据集与表空间对应的数据文件拷贝到目的数据库。

此时可以使用任何传输二进制数据的方式，比如使用 FPT 等工具。我们将该数据放在目的数据库的 D:\test2 目录下。再将表空间的元数据和数据文件导入目的数据库 ORCL1 中，如下例所示。

例子 9-26 将表空间的元数据和数据文件导入目的数据库 ORCL1。

```
D:\>set oracle sid=orcl1
D:\>imp  sys/oracle  file=d:\test2\expdp.dmp  datafiles=d:\test2\test2.dbf
transport_tablespace=y
```

```
Import: Release 10.2.0.1.0 - Production on 星期五 7月 2 11:44:06 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


IMP-00058: 遇到 ORACLE 错误 28009
ORA-28009: connection as SYS should be as SYSDBA or SYSOPER 用户名: sys as sysdba
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.02.01 创建的导出文件
即将导入可传输的表空间元数据...
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
. 正在将 SYS 的对象导入到 SYS
. 正在将 SYS 的对象导入到 SYS
. 正在将 SCOTT 的对象导入到 SCOTT
. . 正在导入表                    "TEST2_EMP"
. 正在将 SYS 的对象导入到 SYS
成功终止导入, 但出现警告。
```

在目标数据库端验证表空间 TEST2 是否成功导出到数据库 ORCL1，方法如下例所示。

例子 9-27 验证表空间 TEST2。

```
SQL> select tablespace_name,status ,contents
  2  from dba_tablespaces
  3  where tablespace_name='TEST2';

TABLESPACE_NAME                STATUS    CONTENTS
------------------------------ --------- ---------
TEST2                        READ ONLY PERMANENT
```

输出结果说明表空间 TEST2 已经导入到数据库 ORCL1 了，但是此时的表空间状态 STATUS 依然是 READ ONLY，所以需要将该表空间设置为 READ WRITE 模式，方法如下例所示。

例子 9-28 修改目标数据库的表空间 TEST2 为读写。

```
SQL> alter tablespace test2 read write;

表空间已更改。
```

我们继续查看表空间 TEST2 的状态，如下例所示。

例子 9-29 查看表空间 TEST2 的状态。

```
SQL> select tablespace_name,status ,contents
  2  from dba_tablespaces
  3  where tablespace_name='TEST2';

TABLESPACE_NAME                STATUS    CONTENTS
------------------------------ --------- ---------
```

```
TEST2                          ONLINE    PERMANENT
```

此时的 STATUS 为 ONLINE 说明该表空间已经为读写模式。

04 将数据库 ORCL2 端的传输表空间 TEST2 设置为 READ WRITE 模式，如下例所示。

例子 9-30 修改源数据库的表空间为读写模式。

```
SQL> alter tablespace test2 read write;

表空间已更改。
```

9.4.7 使用 EXP/IMP 实现跨平台表空间迁移（不同字节序列）

在不同平台之间迁移表空间关键是确认两个平台的 ENDIANESS 是否相同，如果不同则需要进行转换，其他步骤与同平台的表空间迁移类似。这里我们给出一个完整的例子，源数据库在 WINDOWS 平台，表空间为 TEST2 数据文件为 TEST.DBF，而目标数据库运行在 Solaris 平台。

不同 OS 平台表空间迁移流程如图 9-2 所示。

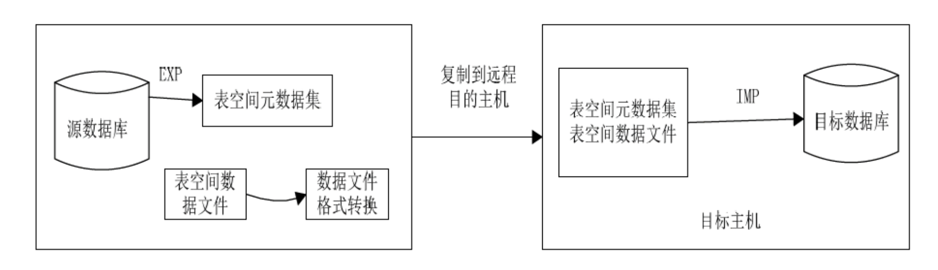


图 9-2　不同 OS 平台表空间迁移流程

其实相同操作系统平台之间与异构操作系统平台之间的表空间迁移的主要区别就是数据文件格式转换，将源数据库平台的字节序列转换为目标数据库所在操作系统平台的字节序列。下面是跨操作系统的表空间迁移步骤。

01 确认两个操作系统平台上的 ENDIANESS 以及是否支持跨平台传输表空间。

首先在 WINDOWS 平台上确认，如下例所示。

例子 9-31 在源数据库端查看操作系统平台以及字节序列格式。

```
SQL> col platform_name for a30
SQL> SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
  2  FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
  3* WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME

PLATFORM_NAME                          ENDIAN_FORMAT
------------------------------ --------------------------
Microsoft Windows IA (32-bit)              Little
```

如上例的查询返回一行，说明当前的数据库支持跨平台传输，而且当前的数据库操作系统平

台为 Microsoft Windows IA (32-bit)，ENDIAN_FORMAT 为 LITTLE_ENDIAN。

下面我们继续在 Solaris 系统上做同样的查询，如下例所示。

例子 9-32 在目标数据库端查看操作系统平台以及字节序列格式。

```
SQL> col platform_name for a30
SQL> SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
  2  FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
  3* WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME

PLATFORM_NAME                          ENDIAN_FORMAT
---------------------------- --------------------------
Solaris[tm] OE (32-bit)                    Big
```

输出说明目标操作系统平台为 Solaris[tm] OE (32-bit)，而该平台的字节序列格式为 Big，与 WINDOWS 平台相比，二者的字节序列格式不同，所以需要进行格式转换。

02 检查表空间是否自包含，如下面例子所示。

例子 9-33 表空间 TEST2 的自包含检查。

```
SQL> conn sys/oracle as sysdba
已连接。
SQL> execute dbms_tts.transport_set_check('test2',true);

PL/SQL 过程已成功完成。
```



如果上述有多个表空间需要验证，则使用如下格式书写需要验证的多个表空间。执行如下指令：

```
EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('sales_1,sales_2', TRUE);
```

接着，我们使用数据字典 TRANSPORT_SET_VIOLATIONS 来查看是否有违反自包含特性。

例子 9-34 查看是否违反自包含特性。

```
SQL> select *
  2  from transport_set_violations;

未选定行
```

如果在数据字典 transport_set_violations 中记录了违反自包含的数据库对象，则修改这些问题。然后继续下一步。



执行过程 TRANSPORT_SET_CHECK 需要用户具备 EXECUTE_CATALOG_ROLE 角色，默认该角色赋予了 SYS，我们就是在 SYS 用户下进行的表空间自包含验证。如果使用其他用户模式下执行自包含检查，需要赋予该用户 EXECUTE_CATALOG_ROLE 角色。

03 将表空间设置为 READ ONLY 并备份表空间的元数据集，如下例所示。

例子 9-35 将表空间 TEST2 设置为只读模式。

```
SQL> alter tablespace test2 read only;

表空间已更改。
```

接着就可以备份表空间 TEST2 的元数据集，如下例所示。

例子 9-36 导出表空间 test2 的元数据。

```
D:\>exp sys/oracle file=expdp.dmp tablespaces=test2 transport_tablespace=y

Export: Release 10.2.0.1.0 - Production on 星期五 7月 2 11:34:46 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


EXP-00056: 遇到 ORACLE 错误 28009
ORA-28009: connection as SYS should be as SYSDBA or SYSOPER
用户名: sys as sysdba
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集
注: 将不导出表数据 (行)
即将导出可传输的表空间元数据...
对于表空间 TEST2...
. 正在导出簇定义
. 正在导出表定义
. . 正在导出表                    TEST2 EMP
. 正在导出引用完整性约束条件
. 正在导出触发器
. 结束导出可传输的表空间元数据
成功终止导出, 没有出现警告。
```

从输出可以看出，导出的是表空间中数据库对象的定义，如表的定义、引用完整性、触发器等。

04 转换传输表空间元数据集的 ENDIANESS。

因为是在不同的 OS 平台上进行传输表空间，而 WINDOWS 平台和 Solaris 平台的 ENDIAN_FORMAT 不同，所以需要进行转换。根据第一步确认的目标数据库 ENDIAN_FORMAT 的格式进行转换。该转换既可以在源数据库端，也可以在目标数据库端实现。我们在源数据库端进行转换，首先打开 RMAN，如下例所示。

例子 9-37 打开 RMAN。

```
D:\>set oracle_sid=orcl1

D:\>rman

恢复管理器: Release 10.2.0.1.0 - Production on 星期二 7月 6 11:04:06 2010
```

```
Copyright (c) 1982, 2005, Oracle. All rights reserved.

RMAN> connect target sys/oracle

连接到目标数据库: ORCL1 (DBID=1107743722)

RMAN>
```

我们在源数据库端进行数据文件格式转换，将表空间 TEST2 的数据文件的 ENDIANNESS 转换为 BIG，使得适应 Solaris[tm] OE (32-bit)操作系统平台，如下例所示。

例子 9-38 在源数据库端转换表空间 TEST2 的数据文件字节序列格式。

```
RMAN> convert tablespace test2
2> to platform 'Solaris[tm] OE (32-bit)'
3> format 'd:/expdp trans.dmp';

启动 backup 于 06-7 月 -10
使用通道 ORA DISK 1
通道 ORA DISK 1: 启动数据文件转换
输入数据文件 fno=00005 name=D:\TEST2\TEST2.DBF
已转换的数据文件 = D:\EXPDP TRANS.DMP
通道 ORA DISK 1: 数据文件转换完毕，经过时间: 00:00:03
完成 backup 于 06-7 月 -10
```

在上述转换中，表空间 TEST2 的所有数据文件完成了格式转换，使得该数据文件支持 Solaris[tm] OE (32-bit)平台支持的字节序列存储格式 BIG，并将转换后的数据文件保存为 D:\EXPDP_TRANS.DMP 文件。

05 将导入备份的表空间元数据集以及转换后的数据文件拷贝到目标数据库。

此时可以使用任何二进制的数据传输方法，如使用 FTP、操作系统的 COPY 工具，RMAN 工具等传输数据，其他方法方法方法方法。

06 在目标数据库端执行表空间的导入，如下例所示。

例子 9-39 使用 IMP 工具将在目的数据库端导入 TEST2 表空间。

```
[root@oracle$] imp sys/oracle file=/u01/bk/expdp.dmp datafiles=/u01/bk
/expdp trans.dmp
transport_tablespace=y
```

在上例中，参数 FILE 表示要导入的表空间元数据集，这个数据集是我们在源数据库端备份过的。datafiles 是表空间 TEST2 对应的数据文件，这个数据文件时经过转换后的数据文件，此时的数据文件支持系统 Solaris[tm] OE (32-bit)。

如果没有在源数据库端进行传输表空间集的转换，就需要在目标数据库端进行传输表空间数据文件的格式转换，即字节序列的转换，此时的数据文件必须传输到目标数据库，然后执行如下的转换。

例子 9-40 在目标数据库端进行数据文件格式转换。

```
RMAN> CONVERT DATAFILE
2> '/u01/finance/orabackup/test2.dbf',
```

```
5> TO PLATFORM="Solaris[tm] OE (32-bit)"
6> FROM PLATFORM=" Microsoft Windows IA (32-bit)"
7> DB_FILE_NAME_CONVERT=
8> "/u01/finance/orabackup/", "/u01/finance/oradata/ "
9> PARALLELISM=5;
```



上例中，把数据文件拷贝到目标数据库目录/u01/finance/orabackup/下，数据文件名是test2.dbf，TO PLATFORM 说明要转换的目标数据库平台，FROM PLATFORM 说明要源数据库操作系统平台版本，DB_FILE_NAME_CONVERT 说明将源数据文件目录转化后存储到目标目录。

9.5　本章小结

本章我们介绍了 EXP/IMP 备份与恢复数据库工具，虽然它是较老的数据库备份工具，但是由于 Oracle 新版本的兼容特性，当前的数据库版本依然支持该特性。使用 EXP 实现对数据库的逻辑备份，可以指定对全库、某个数据库模式、特定的表空间以及特定的表进行备份；相应地使用 IMP 指令对不同的数据库级别进行恢复。

最后，我们介绍了如何使用 EXP/IMP 完成表空间的传输，传输表空间相比导入导出工具能更快速地进行数据迁移。我们不介绍了传输表空间的优势以及局限，还给出一个详细的例子说明如何在相同操作系统平台和异构操作系统平台完成表空间的数据迁移。

第 10 章

◂Oracle数据泵技术▸

上一章我们讲述了如何使用 EXP 和 IMP 实用程序备份和恢复数据库，但是在 Oracle 11g 及以上版本中，建议使用数据泵来代替 EXP 和 IMP 实用程序。本节我们将讲述如何使用数据泵技术特性，以及如何使用 Oracle 11g 的数据泵技术导入和导出数据。

在 Oracle 11g 中使用数据泵技术实现数据的导出和导入数据库。数据泵技术提供了许多新的特性如可以中断导出\导入作业再恢复作业的执行、从一个会话中监控数据泵取作业、在作业执行过程中修改作业属性，以及重启一个失败的数据泵取作业等。

10.1 数据泵导入导出简介

10.1.1 数据泵导入导出技术的结构

在数据泵导入导入技术中，涉及导入实用程序 expdp 和导出实用程序 impdp。当启动数据泵导入或导出程序时，在数据库服务器端启动相应的服务器进程，完成数据的导入及导出任务，所以我们也称数据泵技术是基于 Oracle 数据库服务器的，导入及导出的数据文件也保存在数据库服务器端。为了说明数据泵导入及导出的结构，我们给出示意图说明，如图 10-1 所示。

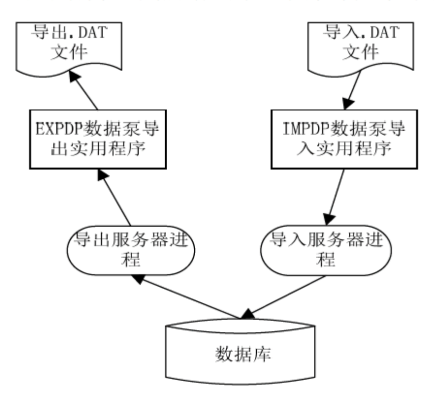


图 10-1　数据泵导入导出结构图

传统的 EXP 实用程序是一个普通的本地用户进程，它将备份的数据写入本地磁盘空间。EXP 实用程序是普通会话的一部分，它从服务器进程中获得要备份的数据。而数据泵取即 EXPDP 程序

启动数据库服务器端的服务器进程，服务器进程完成数据的备份，并将备份文件写入数据库服务器端的计算机磁盘空间，文件格式是 filename.dat。导出的备份文件在导入时只能通过数据泵的导入实用程序 IMPDP 完成，将数据导入到运行在其他平台上的数据库中。



在使用数据泵取技术导出备份数据时，只能将备份的数据写入磁盘文件，而无法写入磁带设备。如果需要写入磁带设备，必须通过第三方的工具实现。

10.1.2　数据泵导入导出技术的优点

在 Oracle 11g 中，Oracle 更新了 EXP/IMP 实用程序，对其进行了功能扩展，实现了这些实用程序的更新版本，即数据泵导入和导出技术。数据泵技术同样由两部分组成：数据泵导出程序（EMPDP）和数据泵导入程序（IMPDP），前者从数据库备份数据，后者从备份文件恢复到数据库中。通过调用 EXPDP 和 IMPDP 启动这两个数据泵导出和导入实用程序。数据泵技术基于数据库服务器，而传统的 EXP/IMP 实用程序基于客户机运行。

使用数据泵技术的优点说明如下。

- 导入导出速度更快：因为在数据泵导入导出作业中可以启动多个线程，所以可以并行实现作业。对于移动大数据量，性能显著提高。
- 重启失败的作业：这个功能是传统的 EXP/IMP 程序无法实现的，不论是数据泵导入导出作业停止还是失败，都可以很容易的重启作业。同时也支持手动停止或重启作业。
- 实时交互能力：在一个运行的数据泵作业中，可以从其他屏幕或控制终端与当前数据泵作业交互，以监控作业的执行以及更改作业的某些参数。
- 独立于客户机：因为数据泵技术是基于数据库服务器的，它是数据库服务器的一部分，一旦启动数据泵作业，则与客户机无关。
- 支持网络操作：支持在两个联网的数据库服务器之间导入和导出数据文件，也支持直接将数据从一个数据库导入另一个数据库，而不需要备份文件。网络操作的方式基于数据库连接，在数据库间直接移动数据的方法不需要磁盘存储。
- 导入功能更加细粒度：在数据泵技术中，使用 INCLUDE 和 EXCLUDE 参数使得数据泵实用程序可以导入或导出更加细粒度的对象，如可以选择只导出过程或函数等。
- 支持增量备份以及快速增量备份，使得备份的数据量减少，加快了备份速度。

10.1.3　数据泵导入导出的目录对象

数据泵作业在数据库服务器上创建所有的备份文件，而 Oracle 要求数据泵必须使用目录对象，以防止用户误操作数据库服务器上特定目录下的操作系统文件。目录对象对应于操作系统上的一个指定目录。

如果当前用户是 DBA 用户，可以使用默认的目录对象而不必再创建数据泵操作的工作目录。此时，数据泵作业会将备份文件、日志文件以及 SQL 文件存储在该目录下，我们使用如下指令查找该默认目录。

例子 10-1　使用数据字典 DBA_DIRECTORIES 查询数据泵作业的目录对象。

```
SQL> conn system/oracle@orcl
已连接。
SQL> col directory name for a15
SQL>col owner for a10
SQL>col directory path for a50
SQL> select *
  2  from dba directories
  3  where directory name ='DATA PUMP DIR';

OWNER      DIRECTORY NAME  DIRECTORY PATH
--------------- ------------------ ------------------------------
SYS        DATA_PUMP_DIR   F:\app\Administraor\admin\orcl\dpdump\
```

使用 SYSTEM 用户登录数据库，查找目录名为 DATA_PUMP_DIR 对应的操作系统文件。该文件对应的目录就是 DBA 用户使用数据泵导出数据时的存储目录。

如果用户欲使用 EXPDP 或者 IMPDP 但是没有可用的目录可用，也不具备创建目录的权限，则提示错误，说明 Oracle 找不到目录对象，无法启动数据泵作业如下例所示。

例子 10-2　不具备目录对象的数据泵作业错误。

```
D:\ expdp scott/tiger@orcl

Export: Release11.1.0.6.0 - Production on 星期四, 27 8 月, 2009 18:45:08

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
ORA-39002: 操作无效
ORA-39070: 无法打开日志文件。
ORA-39145: 必须指定目录对象参数且不能为空
```

如果用户需要自己创建目录对象，需要具有 CREATE ANY DIRECTORY 权限，如下例所示。首先向 SCOTT 用户授权 CREATE ANY DIRECTORY，然后创建属于 SCOTT 用户的数据泵目录对象。

例子 10-3　向 SCOTT 用户授权并创建目录对象。

```
SQL> conn system/oracle@orcl
已连接。
SQL> grant create any directory to scott;

授权成功。

SQL> conn scott/tiger@orcl
已连接。
SQL> create directory scott_pump_dir as 'f:/scottpumpdir';

目录已创建。
```

下面我们在数据库服务器上创建数据泵导入导出目录，此时采用 SYSTEM 用户登录，创建目

录对象 PUMP_DIR，它对应的操作系统目录为 f:\pump，目的是为下节的使用数据泵实现数据导出做准备。创建过程如下例所示。

例子 10-4 创建数据泵导入导出目录。

```
SQL> create directory pump dir as 'f:\pump';

目录已创建。
```

此时，我们创建了一目录。该目录可以给其他用户使用，但是必须将读、写该目录的权限赋予用户。我们可以将该目录的读、写权赋予 SCOTT 用户，如下例所示。

例子 10-5 授予用户 SCOTT 对目录 pump_dir 写的权限。

```
SQL> grant read on directory pump_dir to scott;

授权成功。
```

一旦授权成功，在使用数据泵导入或导出 SCOTT 用户的数据时，就可以使用该目录作为存储和恢复目录了。

数据泵导出 EXPDP 类似于传统的 EXP 实用程序导出数据，使用 EXPDP 允许挂起和恢复作业，并且实现与正在运行的作业交互，可以从正在运行的导出作业中分离，也支持从失败或终止的作业中重启。

10.2 数据泵导入导出与EXP/IMP技术的区别

我们已经提到，Oracle 数据泵技术是对传统的 EXP 和 IMP 实用程序的扩展，使得在数据库服务器端快速的移动数据。这里我们给出数据泵技术和 EXP/IMP 技术的主要区别，读者可以在使用时根据需要进行取舍。

- 数据泵技术比传统的 EXP/IMP 技术能更快速地移动大量数据，因为数据泵技术采用并行流技术实现快速的并行处理。
- 数据泵技术基于数据库服务器。在启动数据泵导入导出实用程序时，在数据库服务器端产生服务器进程负责备份或导入数据，并将备份的数据备份在数据库服务器端，而且服务器进程与 EXPDP 客户机建立的会话无关。
- 传统的 EXP/IMP 是类似于普通的用户进程，执行象 SELECT、INSERT、CREATE 等的 SQL 语句。而数据泵技术类似于启动作业的控制进程，不但启动客户端进程建立会话，还控制整个导入或导出过程，如重启作业。
- 使用传统的 EXP/IMP 实用程序导出的数据格式与数据泵技术导出的数据格式不兼容。
- 数据泵技术与传统的导入导出实用程序不同，它使用目录和目录对象存储数据泵导出文件。使用数据泵导出数据前，必须先创建目录对象，否则无法使用数据泵导入和导出作业。

10.3　数据泵导出（EXPDP）数据库实例

本节我们介绍数据泵导出 EXPDP 的使用方法，主要是介绍参数的含义，然后通过具体的实例演示如何使用 EXPDP 导出整个数据库、导出特定用户以及特定的表空间和数据文件。

10.3.1　数据泵导出的参数含义

使用 EXPDP 使得导出时对象选择的粒度更细，并提供并行处理，使用多个数据流导出数据，允许控制导出操作使用的线程数。EXPDP 支持两种数据访问方法，即使用外部表和直接路径访问方法。使用外部表允许数据库服务器从操作系统文件中读取数据，而直接路径方法使用直接路径 API，它很好地改善了导出导入的性能，因为其内部流的数据格式和存储在备份文件中的数据格式相同，减少了数据转换的时间。

EXPDP 提供了三种提取数据的方法，一是只提取数据库中的元数据即数据库对象的定义，二是只提取数据库中的数据而忽略数据库对象的定义，三是同时提取数据库中的元数据和数据。

在数据泵操作过程中，使用 CTRL+C 组合使将数据泵操作在后台运行，再将 EXPDP 设置为交互模式，此时后台管理运行的 EXPDP 作业，用户可以使用同一窗口进行其他操作。如果需要可以切换回到 EXPDP 作业。为了管理方便，每一个作业都指定了作业名 JOB_NAME，Oracle 使用这个作业名跟踪作业或重新连接该作业。也可是使用默认作业名，在导出作业完成时会给出该名称。

下面我们分析一下，EXPDP 实用程序的参数，如下例所示。

例子 10-6 EXPDP 实用程序参数。

```
D:\>expdp help=y

Export: Release 11.1.0.6.0 - Production on 星期四, 27 8 月, 2009 19:14:22
Copyright (c) 1982, 2007, Oracle.  All rights reserved.

数据泵导出实用程序提供了一种用于在 Oracle 数据库之间传输
数据对象的机制。该实用程序可以使用以下命令进行调用:

   示例: expdp scott/tiger DIRECTORY=dmpdir DUMPFILE=scott.dmp

您可以控制导出的运行方式。具体方法是: 在 'expdp' 命令后输入
各种参数。要指定各参数, 请使用关键字:

   格式:  expdp KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)
   示例: expdp scott/tiger DUMPFILE=scott.dmp DIRECTORY=dmpdir SCHEMAS=scott
            或 TABLES=(T1:P1,T1:P2), 如果 T1 是分区表

USERID 必须是命令行中的第一个参数。

关键字                  说明 (默认)
--------------------------------------------------------------------
ATTACH                  连接到现有作业, 例如 ATTACH [=作业名]。
COMPRESSION             减小有效的转储文件内容的大小
                        关键字值为: (METADATA_ONLY) 和 NONE。
```

```
CONTENT                  指定要卸载的数据，其中有效关键字为:
                         (ALL), DATA_ONLY 和 METADATA_ONLY。
DIRECTORY                供转储文件和日志文件使用的目录对象。
DUMPFILE                 目标转储文件 (expdat.dmp) 的列表,
                         例如 DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp。
ENCRYPTION PASSWORD          用于创建加密列数据的口令关键字。
ESTIMATE                     计算作业估计值，其中有效关键字为:
                             (BLOCKS) 和 STATISTICS。
ESTIMATE ONLY                在不执行导出的情况下计算作业估计值。
EXCLUDE                      排除特定的对象类型，例如 EXCLUDE=TABLE:EMP。
FILESIZE                     以字节为单位指定每个转储文件的大小。
FLASHBACK SCN                用于将会话快照设置回以前状态的 SCN。
FLASHBACK TIME               用于获取最接近指定时间的 SCN 的时间。
FULL                         导出整个数据库 (N)。
HELP                         显示帮助消息 (N)。
INCLUDE                      包括特定的对象类型，例如 INCLUDE=TABLE DATA。
JOB NAME                     要创建的导出作业的名称。
LOGFILE                      日志文件名 (export.log)。
NETWORK LINK                 链接到源系统的远程数据库的名称。
NOLOGFILE                    不写入日志文件 (N)。
PARALLEL                     更改当前作业的活动 worker 的数目。
PARFILE                      指定参数文件。
QUERY                        用于导出表的子集的谓词子句。
SAMPLE                       要导出的数据的百分比;
SCHEMAS                      要导出的方案的列表 (登录方案)。
STATUS                       在默认值 (0) 将显示可用时的新状态的情况下,
                             要监视的频率 (以秒计) 作业状态。
TABLES                       标识要导出的表的列表 - 只有一个方案。
TABLESPACES                  标识要导出的表空间的列表。
TRANSPORT FULL CHECK         验证所有表的存储段 (N)。
TRANSPORT TABLESPACES        要从中卸载元数据的表空间的列表。
VERSION                      要导出的对象的版本，其中有效关键字为:
                             (COMPATIBLE), LATEST 或任何有效的数据库版本。

下列命令在交互模式下有效。
注: 允许使用缩写

命令                         说明
------------------------------------------------------------------------------
ADD_FILE                     向转储文件集中添加转储文件。
CONTINUE_CLIENT              返回到记录模式。如果处于空闲状态，将重新启动作业。
EXIT_CLIENT                  退出客户机会话并使作业处于运行状态。
FILESIZE                     后续 ADD_FILE 命令的默认文件大小 (字节)。
HELP                         总结交互命令。
KILL_JOB                     分离和删除作业。
PARALLEL                     更改当前作业的活动 worker 的数目。
                             PARALLEL=<worker 的数目>。
START_JOB                    启动/恢复当前作业。
STATUS                       在默认值 (0) 将显示可用时的新状态的情况下,
                             要监视的频率 (以秒计) 作业状态。
                             STATUS[=interval]
STOP_JOB                     顺序关闭执行的作业并退出客户机。
                             STOP_JOB=IMMEDIATE 将立即关闭
```

```
                    数据泵作业。
```

在上述输出中，参数的解释已经很详细，一旦读者尝试过使用 EXPDP 导出数据，就很容易掌握和理解这些参数。为了使读者更清晰地理解参数的含义，下面我们解释一些经常使用的 EXPDP 参数指令。

- ATTACH：说明 EXPDP 附加到一个正在运行的、现有的 EXPDP 作业。方式为 ATTACH = JOB_NAME。
- CONTENT：说明要导出的数据是元数据还是数据，或者包括元数据和数据，选项包括 ALL、DATA_ONLY 和 METADATA_ONLY。
- DIRECTORY：说明要导出的备份文件、日志文件和 SQL 文件的存储目录，此时必须事先创建该目录对象。当然可以将其他用户创建的目录对象赋予该当前用户。否则无法启动 EXPDP 程序。
- DUMPFILE：导出的备份文件的文件名，格式为 FILENAME.DMP。我们给出一个例子如下所示。

```
D:\>exp system/oracle@orcl dumpfile =system_dataonly.dmp content =data_only
```

- ESTIMATE：计算 EXPDP 导出作业的导出文件的大小，选项包括基于 BLOCKS 或者基于 STATISTICS，其中 BLOCKS 基于数据库块大小的倍数计算备份文件大小，而基于 STATISTICS 使用当前对象的统计量来计算导出的备份文件的大小，如下例所示。

```
正在使用 STATISTICS 方法进行估计...
处理对象类型 DATABASE EXPORT/SCHEMA/TABLE/TABLE DATA
ORA-39119: 数据泵不支持 XMLSchema 对象。将跳过
TABLE DATA:"OE"."PURCHASEORDER"。
.  预计为 "SH"."CUSTOMERS"                        9.540 MB
............
.  预计为 "SYSTEM"."SQLPLUS PRODUCT PROFILE"            0 KB
.  预计为 "TSMSYS"."SRS$"                           0 KB
使用 STATISTICS 方法的总估计: 49.08 MB
```

- ESTIMATE_ONLY：在 EXPDP 没有实际导出作业前估计导出文件的大小，该参数的值为 Y 或 N，如下例所示。

```
D:\>expdp system/oracle@orcl directory = pump dir estimate only=y

Export: Release 11.1.0.6.0 - Production on 星期四, 27 8月, 2009 20:33:10

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
启动 "SYSTEM"."SYS EXPORT SCHEMA 01":  system/********@orcl directory =
pump dir estimate only=y
正在使用 BLOCKS 方法进行估计...
处理对象类型 SCHEMA EXPORT/TABLE/TABLE DATA
.  预计为 "SYSTEM"."SQLPLUS PRODUCT PROFILE"            0 KB
............
使用 BLOCKS 方法的总估计: 17.37 MB
```

```
作业 "SYSTEM"."SYS_EXPORT_SCHEMA_01" 已于 20:33:20 成功完成
```

- EXCLUDE：排除不需要导出的特定对象类型，如 INCLUDE=TABLE:DEPT，对于任何不导出的对象，也不会导出与它有依赖关系的对象，如不导出表，也不会导出和表相关的任何索引、过程和约束等。
- FLASHBACK_SCN：允许在导出数据库时使用数据库闪回特性，此时 EXPDP 使用规定的 SCN 进行闪回。
- FULL：说明是否导出整个数据库对象，如果该参数为 Y，说明导出数据库的所有对象。
- INCLUDE：说明要导出的特定对象类型，此时会导出该参数指定的对象和与它们有依赖关系的对象。
- JOB_NAME：为了便于管理运行的 EXPDP 作业，设置当前作业的名字。系统默认的命名格式为 sys_operation_mode_nn。如导出 SCOTT 用户的元数据，此时的作业名字为 "SCOTT"."SYS_EXPORT_SCHEMA_01"。
- LOGFILE：说明在导出操作时记录导出过程的日志文件名，其默认名为 export.log，和导出文件保存在相同的目录下，即 directory 参数指定的目录。
- PARALLEL：说明在导出作业时最大的线程数，实现导出作业的并行处理。也可以在作业运行时使用 ATTACH 改变并行度。PARALLEL 参数的默认值为 1，表示使用单线程导出单独个备份文件，如果设置多个工作线程，则要指定相同数量的备份文件，这样多个线程可以同时写多个备份文件。给出一个例子如下所示，设置并行度为 2。

```
D:\>expdp system/oracle@orcl directory =pump_dir
 dumpfile =(para_exp01.dmp,para_exp02.dmp) parallel =2
```

在上例中，我们使用逗号分隔符设置了备份文件的列表，此时并行度为 2，EXPDP 进程会并行对两个文件备份数据。

其实，也可以采用替换变量%U 的方式，说明应该创建多个备份文件，如下例所示。

```
D:\> D:\>expdp system/oracle@orcl directory=pump_dir
dumpfile = para_export%u.dmp parallel = 3
```

采用替换变量创建多个备份文件名时，文件名的格式为 para_exportNN.dmp。上例将产生 3 个备份文件分别为 para_export01.dmp、para_export02.dmp、para_export03.dmp。

- QUERY：允许使用 SQL 语句程序过滤导出的数据，在 Oracle11g 中，允许使用表名限定 SQL 语句，使得 SQL 语句适用于特定的表，如下例所示。
- QUERY=SCOTT.EMP:"WHERE SAL>3000"。说明表 EMP 中工资 SAL 大于 3000 的表被导出。
- SCHEMAS：说明要导出数据的模式。该模式列表可以有多个，使用逗号隔开，如果登录的用户不是导出数据的模式，则登录用户必须拥有 exp_full_database 的权限。
- STATUS：该参数在给定的时间间隔内给出作业的状态。该参数以秒为单位，默认值为 0。如下例所示。

```
D:\>expdp system/oracle@orcl full=y status = 5 directory = pump dir
dumpfile=system full stauts1.dmp
…………
```

```
作业: SYS_EXPORT_FULL_01
  操作: EXPORT
  模式: FULL
  状态: EXECUTING
  处理的字节: 0
  当前并行度: 1
  作业错误计数: 0
  转储文件: F:\PUMP\SYSTEM_FULL_STAUTS.DMP
写入的字节: 4,096
Worker 1 状态:
  状态: EXECUTING
正在使用 BLOCKS 方法进行估计...
…………
Worker 1 状态:
  状态: EXECUTING
  对象类型:
DATABASE EXPORT/SCHEMA/TABLE/POST INSTANCE/PROCACT INSTANCE
  完成的对象数: 18
  总的对象数: 18
  Worker 并行度: 1
处理对象类型 DATABASE EXPORT/SCHEMA/TABLE/POST INSTANCE/PROCDEPOBJ

作业: SYS EXPORT FULL 01
  操作: EXPORT
  模式: FULL
  状态: EXECUTING
  处理的字节: 0
  当前并行度: 1
  作业错误计数: 0
  转储文件: F:\PUMP\SYSTEM FULL STAUTS1.DMP
    写入的字节: 4,096
```

- TABLES：说明要导出数据库表的列表。此时也会导出与表有依赖关系的对象。
- TABLESPACES：说明要导出的数据库表空间的列表。同时会导出其他表空间中与这些表空间中的表有依赖关系的所有对象。
- VERSION：说明要导出的数据库对象到特定版本的数据库。该参数很好的解决了数据库的对象从高版本迁移到低版本过程中版本的兼容问题。

下面解释交互式命令，数据泵技术的交互方式在正在运行的作业中有效，可以使用交互式命令挂起作业，修改作业参数。交互式命令如下例所示。

- ADD_FILE：向导出备份文件集中增加文件以增加目录空间。如在一个作业运行期间输入 CTRL+C 组合键切换到交互式导出提示 EXPORT>。如果该作业因为备份文件的空间不足导致停止，可以使用 ADD_FILE 命令增加文件到导出目录中，指令例子如下。

```
Export>add_file = data_dump_dir:expdata02.dmp;
```

- STOP_JOB：停止运行的数据泵作业，数据库服务器端的导出数据服务器进程终止。

一旦停止一个作业，那么如何重新启动一个数据泵作业呢，此时需要 ATTACH 命令，如完成 SYSTEM 用户的全库导出，采用默认的作业名，该名字在执行全库导出时会提示。停止该作业后，

重新打开这个作业的方法如下例所示。

```
D:\>expdp system/oracle@orcl attach=system.SYS EXPORT FULL 04

Export: Release 11.1.0.6.0 - Production on 星期四, 27 8月, 2009 23:50:59

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options

作业: SYS EXPORT FULL 04
  所有者: SYSTEM
  操作: EXPORT
  创建者权限: FALSE
  GUID: 94DF32382688457CA7085BDFF6A3299E
  起始时间: 星期四, 27 8月, 2009 23:51:00
  模式: FULL
  实例: orcl
  最大并行度: 1
  EXPORT 个作业参数:
  参数名      参数值:
     CLIENT_COMMAND        system/********@orcl full =y directory = pump_dir
dumpfile =export.dmp
  状态: IDLING
  处理的字节: 0
  当前并行度: 1
  作业错误计数: 0
  转储文件: f:\pump\test_jiaohumoshi_backupwholedb_ofsystem1.dmp
    写入的字节: 229,376
```

- START_JOB：重新恢复由于某种意外导致停止的数据泵作业。
- KILL_JOB：杀死客户机进程和数据泵作业（服务器进程）。
- CONTINUE_CLIENT：退出交互方式（EXPORT 方式）恢复正在运行的导出数据泵作业，实际的数据泵作业不受影响。
- EXIT_CLIENT：停止交互式会话并终止客户机会话，但是实际的数据泵作业不受影响。此时用户可以在当前窗口中继续其他操作。

10.3.2 数据泵导出数据库实例

本小节我们通过实例理解如何使用 EXPDP 参数实现数据的导出。使用 EXPDP 可以导出整个数据库、单个模式、特定的表或特定的表空间。以下依次介绍如何实现导出整个数据库或数据库对象。

1. 导出整个数据库

我们使用 SYSTEM 用户登录数据库，限制备份的数据文件的大小为 100M，一旦备份数据文件满，则自动创建一个新的备份文件，使用了替换变量%U 来实现备份文件的自动创建，其中 NOLOGFILE=Y 即不记录备份过程。使用 EXPDP 导出整个数据库的例子如下所示。

例子 10-7 使用 EXPDP 导出整个数据库。

```
    F:\>expdp system/oracle@orcl dumpfile = pump dir:mydb3 %u.dat filesize = 100m
nologfile = y
     job name =tom full = y

    Export: Release 11.1.0.6.0 - Production on 星期六, 22 8月, 2009 10:05:11

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    启动 "SYSTEM"."TOM":  system/********@orcl dumpfile = pump_dir:mydb3_%u.dat
    filesize = 100m nologfile = y job name =tom
    full = y
    正在使用 BLOCKS 方法进行估计...
    处理对象类型 DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
    ORA-39119: 数据泵不支持 XMLSchema 对象。将跳过
    TABLE_DATA:"OE"."PURCHASEORDER"。
    使用 BLOCKS 方法的总估计: 87.5 MB
    处理对象类型 DATABASE_EXPORT/TABLESPACE
    处理对象类型 DATABASE_EXPORT/PROFILE
    ............
    ............
    "TSMSYS"."SRS$"                              0 KB       0 行
    /卸载了主表 "SYSTEM"."TOM"
    ******************************************************************************
     的转储文件集为:
    MYDB3_01.DAT
    "SYSTEM"."TOM" 已经完成, 但是有 1 个错误 (于 10:07:31 完成)
```

上例中备份的文件存储在目录对象 PUMP_DIR 定义的操作系统目录下，此处也可以使用 directory 指令说明目录对象。在 DUMPFILE 参数中指定目录对象使用起来更方便。

2. 导出一个模式

我们导出 SCOTT 模式，在下例中没有 SCHEMA 参数，但是默认导出登录数据库时的模式对象。

例子 10-8 使用 EXPDP 导出一个 SCOTT 模式。

```
    D:\>expdp scott/tiger@orcl dumpfile=pump_dir:scottschema.dmp logfile
    =pump_dir:scottschema.log

    Export: Release 11.1.0.6.0 - Production on 星期五, 28 8月, 2009 0:09:36

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    启动 "SCOTT"."SYS_EXPORT_SCHEMA_01":  scott/********@orcl
    dumpfile=pump dir:scottschema.dmp logfile =pump dir:
    a.log
    正在使用 BLOCKS 方法进行估计...
```

```
处理对象类型 SCHEMA_EXPORT/TABLE/TABLE_DATA
使用 BLOCKS 方法的总估计: 576 KB
处理对象类型 SCHEMA_EXPORT/USER
............
. . 导出了 "SCOTT"."BONUS"                          0 KB       0 行
. . 导出了 "SCOTT"."EMP TEST2"                       0 KB       0 行
已成功加载/卸载了主表 "SCOTT"."SYS_EXPORT_SCHEMA_01"
******************************************************************************
SCOTT.SYS EXPORT SCHEMA 01 的转储文件集为:
  F:\PUMP\SCOTTSCHEMA.DMP
作业 "SCOTT"."SYS_EXPORT_SCHEMA_01" 已于 00:09:52 成功完成
```

3. 导出特定的表

此时使用 TABLES 参数指定导入的表的列表，如果该表不属于登录用户，但是登录用户有访问这些表的权限，则在 TABLES 参数的表必须使用 schema.tablename 的方式。使用 EXPDP 导入特定的表的例子如下所示。

例子 10-9 使用 EXPDP 导入特定的表。

```
F:\>expdp system/oracle@orcl dumpfile = pump dir:scott tables %u.dat
tables=scott.emp,scott.dept nologfile=y job_name=only_scott

Export: Release 11.1.0.6.0 - Production on 星期六, 22 8月, 2009 10:17:33

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
启动 "SYSTEM"."ONLY_SCOTT":  system/********@orcl dumpfile =
pump_dir:scott_tables_%u.dat     tables=scott.emp,scott.dept     nologfile=y
job_name=only_scott
正在使用 BLOCKS 方法进行估计...
处理对象类型 TABLE_EXPORT/TABLE/TABLE_DATA
使用 BLOCKS 方法的总估计: 118 KB
...............
处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
. . 导出了 "SCOTT"."DEPT"                         5.656 KB       4 行
. . 导出了 "SCOTT"."EMP"                          8.390 KB      28 行
已成功加载/卸载了主表 "SYSTEM"."ONLY_SCOTT"
******************************************************************************
SYSTEM.ONLY_SCOTT 的转储文件集为:
  F:\PUMP\SCOTT_TABLES_01.DAT
作业 "SYSTEM"."ONLY_SCOTT" 已于 10:17:42 成功完成
```

4. 导出表空间

导出指定表空间使用 TABESAPCES 参数，如果有多个表空间需要导出，表空间名使用逗号隔开。这里我们使用了 PARALLEL 参数，指定数据导出并行线程数量，与之对应使用替换变量%U 来创建相应数量的备份数据文件，这样每个线程可以独立写一个备份数据文件，提高了导出速度。使用 EXPDP 导出指定的表空间的例子如下所示。

例子 10-10 使用 EXPDP 导出指定的表空间。

```
    D:\>expdp       system/oracle@orcl       dumpfile=pump dir:users tbs %u.dmp
tablespaces=users
    filesize=100m parallel=2 logfile=users tbs.log job name =exp users tbs

    Export: Release 11.1.0.6.0 - Production on 星期五, 28 8月, 2009 0:27:18

    Copyright (c) 1982, 2007, Oracle. All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    启动 "SYSTEM"."EXP_USERS_TBS":  system/********@orcl
    dumpfile=pump dir:users tbs %u.dmp tablespaces=users filesize=100m
    parallel=2 logfile=users_tbs.log job_name =exp_users_tbs
    正在使用 BLOCKS 方法进行估计...
    处理对象类型 TABLE_EXPORT/TABLE/TABLE_DATA
    ORA-39119: 数据泵不支持 XMLSchema 对象。将跳过
    TABLE_DATA:"OE"."PURCHASEORDER"。
    使用 BLOCKS 方法的总估计: 1.611 MB
    . . 导出了 "OE"."LINEITEM_TABLE"                    283.5 KB    2232 行
    ………..
    处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
    已成功加载/卸载了主表 "SYSTEM"."EXP_USERS_TBS"
    ******************************************************************************
    SYSTEM.EXP_USERS_TBS 的转储文件集为:
      F:\PUMP\USERS_TBS_01.DMP
      F:\PUMP\USERS_TBS_02.DMP
    作业 "SYSTEM"."EXP_USERS_TBS" 已经完成, 但是有 1 个错误 (于 00:27:38 完成)
```

5. 只导出数据

使用 EXPDP 的 CONTENT 参数，可以指定导出表数据和元数据（对应参数 ALL），导出表行数据（对应参数 DATA_ONLY），或只导出元数据即表和其他数据库对象的定义（对应参数 METADATA_ONLY）。使用 EXPDP 只导出数据行的例子如下所示。

例子 10-11 使用 EXPDP 只导出数据行。

```
    F:\>expdp    system/oracle@orcl    dumpfile    =pump dir:mydb dataonly %u.dat
filesize= 100m
    job name=larry    full=y    content    =data only    logfile    =pump dir:mydb
exp dataonly log
    Export: Release 11.1.0.6.0 - Production on 星期六, 22 8月, 2009 10:33:09

    Copyright (c) 1982, 2007, Oracle. All rights reserved.
    ;;;
    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    启动 "SYSTEM"."LARRY":  system/********@orcl dumpfile
    =pump dir:mydb dataonly %u.dat filesize=100m job name=larry full=y content
=data only
    logfile =pump dir:mydb exp dataonly log
    正在使用 BLOCKS 方法进行估计...
```

```
    处理对象类型 DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
    ORA-39119: 数据泵不支持 XMLSchema 对象。将跳过
    TABLE_DATA:"OE"."PURCHASEORDER"。
    使用 BLOCKS 方法的总估计: 87.5 MB
    . . 导出了 "SYSTEM"."LIN"                          11.75 MB    8373 行
    …………
    . . 导出了 "TSMSYS"."SRS$"                              0 KB       0 行
    已成功加载/卸载了主表 "SYSTEM"."LARRY"
    ***************************************************************************
****
    SYSTEM.LARRY 的转储文件集为:
      F:\PUMP\MYDB DATAONLY 01.DAT
    作业 "SYSTEM"."LARRY" 已经完成, 但是有 1 个错误 (于 10:34:08 完成)
```

在上例中，我们设置参数 CONTENTS=DATA_ONLY，说明只导出 SYSTEM 用户中表的数据行，参数 LOGFILE 说明要记录导出过程。

6．EXPDP 使用技巧

（1）使用参数文件

在使用 EXPDP 导出数据时，由于参数很多导致每次执行备份都输入一长串指令，这样不但繁琐而且不易修改，Oracle 的数据泵技术允许使用参数文件，用户事先在参数文件中创建各种参数，保存该文件为 praname.par 文件，然后在执行导出时使用参数 PARFILE 指定参数文件的位置执行导出备份，这样就不用输入一长串参数指令。

首先我们建立参数文件如下：

```
    directory =pump dir
    dumpfile=para data only %u.dmp
    content =data only
    exclude =table:"in('salgrade','bonus')"
    logfile=para data only.log
    filesize=50m
    parallel=2
    job_name =para_data_only
```

保存在文件 EXP.PAR 内，然后在使用 EXPDP 备份数据时，可以使用该参数文件，如下例所示。

例子 10-12 使用参数文件备份数据。

```
    D:\>expdp scott/tiger@orcl parfile=exp.par
```

使用参数 PARFILE 说明参数文件的位置必须说明绝对路径。

（2）估计导出文件的空间大小

EXPDP 使用参数 ESTIMATE_ONLY 计算导出数据所需要的存储空间，显然这是个有用的参数，在导出的数据大小不清楚时，事先知道备份文件的大小，可以提前分配磁盘空间，防止由于磁盘空间不足而引起的 EXPDP 导出作业停止。

例子 10-13 使用 EXPDP 导出数据时只计算导出作业所需要的空间。

```
    F:\>expdp  system/oracle@orcl  full  =y  estimate only=y  estimate=statistics
nologfile=y

    Export: Release 11.1.0.6.0 - Production on 星期六, 22 8月, 2009 10:38:11

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    启动 "SYSTEM"."SYS EXPORT FULL 01":  system/********@orcl full =y
    estimate_only=y estimate=statistics nologfi
    正在使用 STATISTICS 方法进行估计...
    处理对象类型 DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
    ORA-39119: 数据泵不支持 XMLSchema 对象。将跳过
    TABLE_DATA:"OE"."PURCHASEORDER"。
    .  预计为 "SH"."CUSTOMERS"                          9.540 MB
    ..........
    .  预计为 "SYSTEM"."SQLPLUS_PRODUCT_PROFILE"             0 KB
    .  预计为 "TSMSYS"."SRS$"                              0 KB
    使用 STATISTICS 方法的总估计: 46.41 MB
    作业 "SYSTEM"."SYS_EXPORT_FULL_01" 已经完成 (于 10:38:46 完成)
```

上述计算过程将使用 STATISTIC 的方法计算 SYSTEM 用户所有数据库对象的大小，最后给出一个总的估计结果。

10.4　数据泵导入（IMPDP）数据库实例

本节我们介绍如何使用 IMPDP 导入数据库，使用 IMPDP 与使用 EXPDP 类似，要求熟悉参数的含义以及使用。我们先介绍 IMPDP 的参数含义以及使用方法，然后给出具体的实例，演示如何导入整个数据库、导入一个表空间、导入特定的表和导入特定数据库对象。

10.4.1　数据泵导入（IMPDP）概述及参数含义

Oracle 数据泵导入实用程序（IMPDP）将备份的数据导入到整个数据库、特定的模式、特定的表或者特定的表空间。使用 IMPDP 也可以在不同平台的数据库之间迁移表空间。IMPDP 实用程序通过各种参数支持对数据过滤以及对元数据的过滤，而元数据的过滤可以有效控制要导入的对象类型，如导入表、索引、授权等等。

使用数据泵导入实用程序与数据泵导出实用程序一样可以使用 DIRECTORY、PARFILE、DUMPFILE 和 LOGFILE 等参数，但是正如 ADD_FILE 是数据泵导出 EXPDP 实用程序专有的一样，SQLFILE 参数也是数据泵导入 IMPDP 实用程序所专有的。在执行数据泵导入作业时，往往需要从备份数据文件中途提取 DDL 语句，此时需要使用 SQLFILE，如下例所示。

例子 10-14 执行数据泵导入作业（使用 SQLFILE 参数）。

```
    D:\>impdp   system/oracle@orcl   directory   =pump_dir   dumpfile=scott_tables
```

```
sqlfile=scott_table.sql

    Import: Release 11.1.0.6.0 - Production on 星期五, 28 8月, 2009 16:43:20

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    已成功加载/卸载了主表 "SYSTEM"."SYS_SQL_FILE_FULL_01"
    启 动  "SYSTEM"."SYS_SQL_FILE_FULL_01":   system/********@orcl  directory
=pump_dir
     dumpfile=scott_tables sqlfile=scott_table.sql
    处理对象类型 TABLE_EXPORT/TABLE/TABLE
    处理对象类型 TABLE_EXPORT/TABLE/INDEX/INDEX
    处理对象类型 TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
    处理对象类型 TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
    处理对象类型 TABLE_EXPORT/TABLE/TRIGGER
    处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
    作业 "SYSTEM"."SYS_SQL_FILE_FULL_01" 已于 16:43:23 成功完成
```

在执行 IMPDP 导入数据时，IMPDP 实用程序会自动从备份文件（DUMPFILE 参数指定文件）中提取 SQL 语句，并把提取的 SQL 语句保存在参数 SQLFILE 指定的文件中。整个导入过程中创建的 scott_table.sql 文件保存在参数 DIRECTORY 指定的目录对象中，SQLFILE 参数的作用是为特定的备份文件提取 SQL 的 DDL 语句，而不是产生实际的 DDL 操作，它提取出备份文件中有关 DDL 的 SQL 脚本内容，使得用户知道在备份文件中发生了那些 DDL 操作。下面我们打开 scott_table.sql 文件，如下所示。

例子 10-15 SCOTT_TABLE.SQL 文件中的部分记录结果。

```
    -- CONNECT SYSTEM
    -- new object type path is: TABLE_EXPORT/TABLE/TABLE
    CREATE TABLE "SCOTT"."DEPT"
       ( "DEPTNO" NUMBER(2,0),
        "DNAME" VARCHAR2(14),
        "LOC" VARCHAR2(11)
       ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 115 NOCOMPRESS LOGGING
      STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
      PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
      TABLESPACE "USERS" ;

    CREATE TABLE "SCOTT"."EMP"
       ( "EMPNO" NUMBER(4,0),
        "ENAME" VARCHAR2(10),
        "JOB" VARCHAR2(9),
        "MGR" NUMBER(4,0),
        "HIREDATE" DATE,
        "SAL" NUMBER(7,2),
        "COMM" NUMBER(7,2),
        "DEPTNO" NUMBER(2,0)
       ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 115 NOCOMPRESS LOGGING
      STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
      PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
```

```
  TABLESPACE "USERS" ;

-- new object type path is: TABLE_EXPORT/TABLE/INDEX/INDEX
-- CONNECT SCOTT
CREATE UNIQUE INDEX "SCOTT"."PK_DEPT" ON "SCOTT"."DEPT" ("DEPTNO")
  PCTFREE 10 INITRANS 2 MAXTRANS 115
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "USERS" PARALLEL 1 ;

  ALTER INDEX "SCOTT"."PK_DEPT" NOPARALLEL;
…………
```

上述的 SQL 脚本显示了导入备份文件过程中要创建表、索引和各种触发器。

数据泵导入 IMPDP 和数据泵导出 EXPDP 一样，需要通过各种参数来控制导入行为，如使用 CONTENT 参数确定是否导入元数据（METADATA_ONLY）、行数据（DATA_ONLY）、还是行数据和元数据（ALL）。DUMPFILE 参数确定导入文件名，LOGFILE 参数确定导入过程的日志文件等。下面我们分类详细介绍各种参数。

使用 IMPDP help=y 指令可以查看 IMPDP 指令的所有可选参数及其基本含义，如下例所示。

例子 10-16　查看 IMPDP 导入程序的所有参数。

```
D:\>IMPDP HELP=Y

Import: Release 11.1.0.6.0 - Production on 星期五, 28 8月, 2009 17:11:22

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


数据泵导入实用程序提供了一种用于在 Oracle 数据库之间传输
数据对象的机制。该实用程序可以使用以下命令进行调用:

    示例: impdp scott/tiger DIRECTORY=dmpdir DUMPFILE=scott.dmp

您可以控制导入的运行方式。具体方法是: 在 'impdp' 命令后输入
各种参数。要指定各参数, 请使用关键字:

    格式:  impdp KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)
    示例: impdp scott/tiger DIRECTORY=dmpdir DUMPFILE=scott.dmp

USERID 必须是命令行中的第一个参数。

关键字                        说明 (默认)
------------------------------------------------------------------------------
ATTACH                       连接到现有作业, 例如 ATTACH [=作业名]。
CONTENT                      指定要加载的数据, 其中有效关键字为:
                             (ALL), DATA_ONLY 和 METADATA_ONLY。
DIRECTORY                    供转储文件, 日志文件和 sql 文件使用的目录对象。
DUMPFILE                     要从 (expdat.dmp) 中导入的转储文件的列表,
                        例如 DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp。
ENCRYPTION_PASSWORD          用于访问加密列数据的口令关键字。
                             此参数对网络导入作业无效。
ESTIMATE                     计算作业估计值, 其中有效关键字为:
```

```
                             (BLOCKS) 和 STATISTICS。
 EXCLUDE                     排除特定的对象类型, 例如 EXCLUDE=TABLE:EMP。
 FLASHBACK SCN               用于将会话快照设置回以前状态的 SCN。
 FLASHBACK TIME              用于获取最接近指定时间的 SCN 的时间。
 FULL                        从源导入全部对象 (Y)。
 HELP                        显示帮助消息 (N)。
 INCLUDE                     包括特定的对象类型, 例如 INCLUDE=TABLE DATA。
 JOB_NAME                    要创建的导入作业的名称。
 LOGFILE                     日志文件名 (import.log)。
 NETWORK LINK                链接到源系统的远程数据库的名称。
 NOLOGFILE                   不写入日志文件。
 PARALLEL                    更改当前作业的活动 worker 的数目。
 PARFILE                     指定参数文件。
 QUERY                       用于导入表的子集的谓词子句。
 REMAP_DATAFILE              在所有 DDL 语句中重新定义数据文件引用。
 REMAP SCHEMA                将一个方案中的对象加载到另一个方案。
 REMAP TABLESPACE            将表空间对象重新映射到另一个表空间。
 REUSE DATAFILES             如果表空间已存在, 则将其初始化 (N)。
 SCHEMAS                     要导入的方案的列表。
 SKIP UNUSABLE INDEXES       跳过设置为无用索引状态的索引。
 SQLFILE                     将所有的 SQL DDL 写入指定的文件。
 STATUS                      在默认值 (0) 将显示可用时的新状态的情况下,
                             要监视的频率 (以秒计) 作业状态。
 STREAMS CONFIGURATION       启用流元数据的加载
 TABLE_EXISTS_ACTION         导入对象已存在时执行的操作。
                             有效关键字: (SKIP), APPEND, REPLACE 和 TRUNCATE。
 TABLES                      标识要导入的表的列表。
 TABLESPACES                 标识要导入的表空间的列表。
 TRANSFORM                   要应用于适用对象的元数据转换。
                             有效的转换关键字: SEGMENT ATTRIBUTES, STORAGE
                             OID 和 PCTSPACE。
 TRANSPORT DATAFILES         按可传输模式导入的数据文件的列表。
 TRANSPORT FULL CHECK        验证所有表的存储段 (N)。
 TRANSPORT TABLESPACES       要从中加载元数据的表空间的列表。
                             仅在 NETWORK LINK 模式导入操作中有效。
 VERSION                     要导出的对象的版本, 其中有效关键字为:
                             (COMPATIBLE), LATEST 或任何有效的数据库版本。
                             仅对 NETWORK LINK 和 SQLFILE 有效。

 下列命令在交互模式下有效。
 注: 允许使用缩写

 命令                        说明 (默认)
 ------------------------------------------------------------------------
 CONTINUE CLIENT             返回到记录模式。如果处于空闲状态, 将重新启动作业。
 EXIT_CLIENT                 退出客户机会话并使作业处于运行状态。
 HELP                        总结交互命令。
 KILL JOB                    分离和删除作业。
 PARALLEL                    更改当前作业的活动 worker 的数目。
                             PARALLEL=<worker 的数目>。
 START JOB                   启动/恢复当前作业。
                             START JOB=SKIP CURRENT 在开始作业之前将跳过
                             作业停止时执行的任意操作。
 STATUS                      在默认值 (0) 将显示可用时的新状态的情况下,
                             要监视的频率 (以秒计) 作业状态。
```

STOP JOB	STATUS[=interval] 顺序关闭执行的作业并退出客户机。 STOP JOB=IMMEDIATE 将立即关闭数据泵作业。

下面我们对 IMPDP 导入程序的参数进行适当分类，使得读者易于理解和掌握，然后再详细介绍这些参数的含义和使用注意事项。

1. 目录和文件相关参数

- DIRECTORY：说明备份文件、日志文件和 SQL 文件的目录对象，如果没有定义目录，则会使用 PUMP_DIR 的默认值。
- DUMPFILE：说明备份文件名，如导入数据时需要多个备份文件，则用逗号分隔这些文件名。在 DUMPFILE 参数后可以使用包括目录，如 DUMFILE=PUMP_DIR:BACKUP.DMP。也可以使用替换变量（%U）告诉 IMPDP 可以使用多个备份文件，如 DUMPFILE=PUMP_DIR:BACKUP_%U.DMP。
- PARFILE：说明参数文件，IMPDP 使用外部定义一个参数文件执行导入行为，该参数文件是本地的，使用时需要告诉 IMPDP 参数文件的绝对位置。如

```
D:\IMPDP SYSTEM/ORACLE@ORCL PARFILE=D:\PAR\EXP.PAR
```

- LOGFILE：说明使用日志文件保存导入过程的信息，该参数的值是日志文件的名字，如 LOGFILE =MYLOG.LOG。
- NOLOGFILE：说明不使用日志文件记录导入过程，如 NOLOGFILE=Y。
- SQLFILE：说明从备份文件中提取 SQL 的 DDL 语句，并写入该参数设置的文件中，如 SQLFILE=MYSQLFILE.SQL。该文件默认保存在 DIRECTORY 参数设置的目录对象中。

2. 过滤参数

- INCLUDE：说明要导入的特定对象，如只导入表，此时会导入与导入特定对象有依赖关系的对象如索引、触发器等。

  下面是使用 INCLUDE 参数的例子，说明只允许导入表对象，且只有两个表可以导入。

```
INCLUDE=TABLE: "IN ('EMP','DEPT') "
```

  也可以使用 QUERY 参数过滤要导入的表数据，此时数据泵导入作业使用外部表数据方法访问数据，而不是采用直接路径方法。如下例所示。

```
INCLUDE=TABLE: "IN ('EMP','DEPT') "
QUERY=EMP: "WHERE sal>3000 ORDER BY sal"
```

- TABLE_EXITS_ACTION：该参数说明当导入的表已经存在时，IMPDP 导入程序的行为，参数 TABLE_EXITS_ACTION 有四个值，SKIP 表示如果该表存在则跳过该表，它是默认值；APPEND 将导入的数据行附加到当前存在的表中；TRUNCATE 截断表并从导入数据文件中重新装载数据；REPLACE 删除存在的表然后重建该表并导入数据。
- EXCLUDE：在导入操作中排除特定的元数据，如不导入特定的表，此时也不会导入和排除对象有依赖关系的其他对象。如下例所示告诉 IMPDP 程序不导入表 EMP 和 DEPT。

```
EXCLUDE=TABLE: "IN ('EMP','DEPT') "
```

3．导入作业参数

- JOB_NAME：说明导入作业名，IMPDP 提供了很多可管理性功能如停止作业和恢复作业、附加（ATTACH）到特定的作业，都需要作业名来关联导入作业。
- PRALLEL：说明当前导入作业的线程数。该值的默认值为 1。
- STATUS：监视导入作业的状态频率。该参数的默认值为 0。

4．导入方式参数

- TABLES：说明允许导入指定的表，如果有多个表使用逗号分隔开，同时也导入与这些表有依赖关系的对象，如索引、触发器和函数等。
- SCHEMAS：说明要导入的模式列表，要使用该参数登录数据库的用户必须拥有 imp_full_database 的权限。
- TABLESPACES：说明要导入的表空间的列表，在导入这些表空间的同时也要求导入与表空间有依赖关系的所有数据库对象。
- FULL：说明要导入整个数据库。该参数的默认值为 n。

5．重新映射参数

重新映射使得在数据导入过程中，将数据从一个数据库对象移动到另一个数据库对象，可以映射模式、映射数据文件和映射表空间，映射可以理解为“数据对象移动”。

- REMAP_SCHEMA：重新映射模式，可以将对象从一个模式移动到另一个模式。

```
D:\>impdp system/oracle@orcl dumpfile =pump dir:SCHEMA SCOTT.DMP
 remap_schema=scott:linzi
```

上例将 SCOTT 模式下的所有数据库对象移动到 LINZI 模式下，这样使用 LINZI 模式登录数据库，就可以使用 SCOTT 用户的所有数据库对象。我们通过下例验证重映射模式的结果。

例子 10-17 验证模式 LINZI 中是否有 SCOTT 模式对象。

```
SQL> conn linzi/linzi@orcl
已连接。
SQL> desc dept;
 名称                                       是否为空？  类型
 ----------------------------------- ---------------- ---------------------
 DEPTNO                                      NOT NULL NUMBER(2)
 DNAME                                                VARCHAR2(14)
 LOC                                                  VARCHAR2(11)
```

可见 SCOTT 用户的 DEPT 表对象在 LINZI 模式下可以直接使用，说明模式迁移成功。

- REMAP_DATAFILE：在导入数据时，重新定义数据文件的名称和目录。如下例所示。

```
D:\impdp  system/oracle@orcl  directory=pump_dir  dumpfile=backup_full.dmp
remap_datafile='c:\mydb.dbf':'d:\mydb\newdb.dbf
```

- REMAP_TABLESPACE：重映射表空间使得将数据对象从一个表空间移动到另一个表空间。如下例所示。

```
D:\impdp system/oracle@orcl remap_tablespace='users':'mynewusers'
```

```
  directory=pump_dir dumpfile=backup_full.dmp
```

6. 转换参数

- TRANSFORM：该参数说明在导入数据泵作业时，可以选择导入某个对象的存储参数或其他属性值，如导入表时，不导入该表的存储属性等。

TRANSFORM 参数的语法如下例所示。

```
TRANSFORM= transform_name:value[:object_type]
```

下面我们介绍这个参数各个部分的用法：

（1）transform_name：转换名由四个选项组成，代表四种基本的对象特征。其中 SEGMENT_ATTRIBUTES 段属性包括物理属性、存储参数、表空间等，该参数的值为 Y 或 N。如果选择 SETMENT_ATTRIBUTES=Y 则说明导入作业包括对象的这些属性。STORAGE 存储属性说明是否导入对象的存储属性，如果 STORAGE=Y 说明对象的存储属性作为导入作业的一部分。OID 说明是否分配新的 OID 给对象表 PCTSPACE：提供一个正数值，可以增加对象的分配尺寸。

（2）value：转换名的值中前三个值即 SEGMENT_ATTRIBUTES、STORAGE 和 OID，默认值为 Y，说明默认数据泵导入对象的存储属性和段属性。而 PCTSPACE 取一个正数值。

（3）object_type：对象类型说明需要转换那些类型的对象，这些类型包括表、索引、表空间以及约束等。

例子 10-18 说明如何使用 TRANSFORM 参数。

```
    D:\>impdp  system/oracle@orcl  tables  =emp  directory  =pump dir  dumpfile=
scott_tables.dmp transform=segment_attributes:n:table

    Import: Release 11.1.0.6.0 - Production on 星期五, 28 8月, 2009 21:07:29

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    已成功加载/卸载了主表 "SYSTEM"."SYS IMPORT TABLE 01"
    启 动   "SYSTEM"."SYS IMPORT TABLE 01":   system/********@orcl  tables  =emp
directory =pump dir dumpfile=scott tables.dmp tra
    nsform=segment attributes:n:table
    处理对象类型 TABLE EXPORT/TABLE/TABLE
    处理对象类型 TABLE EXPORT/TABLE/TABLE DATA
    . . 导入了 "SCOTT"."EMP"                         9.523 KB      56 行
    处理对象类型 TABLE EXPORT/TABLE/TRIGGER
    处理对象类型 TABLE EXPORT/TABLE/STATISTICS/TABLE STATISTICS
    作业 "SYSTEM"."SYS_IMPORT_TABLE_01" 已于 21:07:31 成功完成
```

在上例中，我们事先删除掉 SCOTT 用户的表 EMP，模拟一个故障，此时再使用备份文件 scott_tables.dmp（该文件备份了 SCOTT 用户的 DEPT 和 EMP 表）恢复表 EMP，但是我们不需要该表的 SEGMENT_ATTRIBUTES 属性。

7. 闪回参数

- FLASHBACK_SCN：使用 Oracle 的闪回特性，允许导入和闪回 SCN 接近的数据。

- FLASHBACK_TIME：使用 Oracle 的闪回特性，允许导入与指定闪回时间接近的数据。

8．与可移植表空间有关的参数

Oracle 的可移植表空间使得将数据从一个数据库移动到另一个数据库非常容易，可以方便的将一个数据库表空间中的数据迁移到其他数据库中的表空间中。

- TRANSPORT_TABLESPACES：说明要迁移的表空间列表，如下例所示：

```
D:\>expdp   system/oracle@orcl   directory=pump_dir   dumpfile=users_tbs_
metadata.dmp transport_tablespaces=users
```

要迁移的表空间必须置于只读状态，但是导出备份文件 users_tbs_metadata.dmp 文件只包含表空间 USERS 的元数据。

- TRANSPORT_FULL_CHECK：迁移表空间时，检查迁移表空间内的对象与迁移表空间外的对象是否有依赖性，该参数只有在使用 NETWORK_LINK 参数时才有效。
- TRANSPORT_DATAFILES：在执行表空间导入时，目标数据库将使用源数据库中拷贝过来的数据文件作为可以移植表空间的数据文件。该参数说明数据文件名。

9．交互模式参数

数据泵导入的交互参数和数据泵导出的交互参数功能是一样的，在数据泵导入参数中没有 ADD_FILE 参数，它只对数据泵导出程序有效，而其他参数以及切换到交互模式数据泵导入与导出是一样的。

- PARALLEL：说明当前作业的活跃 WORKER 数量。
- CONTINUE_CLIENT：在切换到交互模式后，返回记录模式。
- EXIT_CLIENT：退出客户登录模式，但是不终止导入作业。
- KILL_JOB：分离或删除当前导入作业。
- START_JOB：在导入作业被意外终止后，可以重启或恢复当前作业。
- STATUS：监视当前导入作业的状态。该参数是一个整数值，默认值为 0；如果设置 STATUS=5，说明每 5 秒钟刷新一次导入作业的状态信息。
- STOP_JOB：关闭当前执行的作业并退出客户端，如果有多个导入作业，则顺序关闭这些作业。如果设置 STOP_JOB=IMMEDIATE 将立即关闭数据泵作业。

10.4.2　数据泵导入（IMPDP）数据库实例

使用数据泵导入 IMPDP 可以导入基于使用数据泵导出的备份文件，可以导入整个数据库、指定的表空间、指定的表或者指定的数据库对象类型，如索引、函数、存储过程和触发器等。下面我们通过实例依次说明如何使用数据泵导入作业。

1．导入整个数据库

导入整个数据库至少需要两个参数。一个是 FULL，设置 FULL=Y 说明是导入全库；另一个是 DUMPFILE，说明要导入的备份文件的目录和名称。当然最好设置 JOB_NAME 参数，因为它允许切换到交换模式，允许终止或重启导入会话，如下例所示。

例子 10-19　使用 IMPDP 导入整个数据库。

```
    D:\>impdp   system/oracle@orcl   dumpfile=pump dir:full db %u.dat   logfile=
myfulldb.log parallel= 3 job_name=my_fulldb_impdp full =y
```

在上例中，我们导入的备份文件位于目录对象 PUMP_DIR 定义的操作系统目录下，这里使用了替换变量%U 说明，它的值为 01~99，IMPDP 程序将依次读取备份文件集中的多个备份文件。参数 LOGFILE=MYFULLDB.LOG 记录数据导入过程，PARALLEL=3 说明启动三个线程完成数据导入，JOB_NAME 为 my_fulldb_impdp。关键是 FULL=Y 说明导入整个数据库，如果不使用 FULL 参数，默认导入模式 SYSTEM 的所有数据库对象。

2. 导入表空间

使用 IMPDP 导入特定的表空间时，需要有备份表空间文件，需要使用 TABLESPACES 参数说明要导入的表空间名，此时实际上是导入表空间中的所有数据库对象，当然这些工作都由 IMPDP 自己操作完成。如果当前的数据库中的表空间已有相应的表对象，则最好告诉 IMPDP 该怎么做，此时需要参数 TABLE_EXITS_ACTION，它的默认值为 SKIP，即如果表已经存在则跳过。我们建议使用 REPLACE 或 TRUNCATE，前者表示重建表，后者表示删除掉当前表中的数据，然后使用备份文件中的表数据进行加载，但是会跳过所有相关元数据，如下例所示。

例子 10-20　使用 IMPDP 导入特定的表空间。

```
D:\>impdp system/oracle@orcl
dumpfile=pump dir:MYDB TBS USERSANDSYSTEM 01.DAT logfile= tablespaces=users
table exists action=replace

Import: Release 11.1.0.6.0 - Production on 星期六, 29 8月, 2009 10:59:40

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
已成功加载/卸载了主表 "SYSTEM"."SYS IMPORT TABLESPACE 01"
启动 "SYSTEM"."SYS IMPORT TABLESPACE 01":  system/********@orcl
dumpfile=pump dir:MYDB TBS USERSANDSYSTEM 01.DAT nologfi
le=y tablespaces=users table exists action=replace
处理对象类型 TABLE EXPORT/TABLE/TABLE
处理对象类型 TABLE EXPORT/TABLE/TABLE DATA
. . 导入了 "LINZI"."CREATE$JAVA$LOB$TABLE"          5.835 KB       1 行
. . 导入了 "OE"."CREATE$JAVA$LOB$TABLE"             5.828 KB       1 行
…………
. . 导入了 "SCOTT"."EMP_TEST2"                          0 KB       0 行
处理对象类型 TABLE_EXPORT/TABLE/INDEX/INDEX
处理对象类型 TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
处理对象类型 TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
处理对象类型 TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
处理对象类型 TABLE_EXPORT/TABLE/TRIGGER
处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
作业 "SYSTEM"."SYS_IMPORT_TABLESPACE_01" 已于 11:00:01 成功完成
```

我们使用了 LOGFILE 参数，这是一个好习惯，因为在备份后可以使用日志文件查询备份过程

和备份的所有数据库对象信息。TABLESPACES 参数是必须的，它说明要导入备份文件中的那个表空间。参数 TABLE_EXISTS_ACTION=REPALCE 说明遇到已经存在的表要重建该表对象然后使用备份文件中的数据进行加载。

3. 导入指定的表

使用 IMPDP 导入特定的表使用 TABLES 参数，该参数后是要导入的表对象的列表，如果有多个表，使用逗号分隔。我们使用了参数 TABLE_EXISTS_ACTION 参数，设置 TABLE_EXISTS_ACTION=REPLACE，如果该表存在则先删除再加载数据，如下例所示。

例子 10-21 导入特定的表对象。

```
    D:\>impdp scott/tiger@orcl dumpfile=pump_dir:MYDB_TBS_USERSANDSYSTEM_01.DAT
nologfile=y tables= emp table_exists_action=replace

    Import: Release 11.1.0.6.0 - Production on 星期六, 29 8月, 2009 11:05:04

    Copyright (c) 1982, 2007, Oracle.  All rights reserved.

    连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
    With the Partitioning, OLAP and Data Mining options
    ORA-39154: 外部方案中的对象已从导入中删除
    已成功加载/卸载了主表 "SCOTT"."SYS_IMPORT_TABLE_01"
    启动 "SCOTT"."SYS_IMPORT_TABLE_01":  scott/********@orcl
    dumpfile=pump_dir:MYDB_TBS_USERSANDSYSTEM_01.DAT nologfile=y ta
    bles= emp table_exists_action=replace
    处理对象类型 TABLE_EXPORT/TABLE/TABLE
    处理对象类型 TABLE_EXPORT/TABLE/TABLE_DATA
    . . 导入了 "SCOTT"."EMP"                         9.523 KB      56 行
    处理对象类型 TABLE_EXPORT/TABLE/TRIGGER
    处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
    作业 "SCOTT"."SYS_IMPORT_TABLE_01" 已于 11:05:08 成功完成
```

上例中，我们导入了表 EMP，注意此时使用 SCOTT 用户登录，所以只导入 SCOTT 用户中的 EMP 表。如果使用 SYSTEM 用户登录就会将 EMP 表导入多个用户，因为 SYSTEM 用户拥有全部数据库权限，如下例所示。

例子 10-22 使用 SYSTEM 用户登录数据库导入 EMP 表。

```
    D:\>impdp system/oracle@orcl
    dumpfile=pump_dir:MYDB_TBS_USERSANDSYSTEM_01.DAT
     nologfile=y tables= emp table_exists_action=replace
    …………
    处理对象类型 TABLE_EXPORT/TABLE/TABLE
    处理对象类型 TABLE_EXPORT/TABLE/TABLE_DATA
    . . 导入了 "LINZI"."EMP"                         9.523 KB      56 行
    . . 导入了 "OE"."EMP"                           9.515 KB      56 行
    . . 导入了 "SCOTT"."EMP"                         9.523 KB      56 行
    处理对象类型 TABLE_EXPORT/TABLE/TRIGGER
    处理对象类型 TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
```

显然此时导入数据时，IMPDP 将在数据库中搜索那些用户拥有 EMP 表，发现当前数据库中用

户 LINZI、OE 和 SCOTT 都拥有该表，所以将它可以“管理”的所有 EMP 表都进行了恢复。

4．导入指定的数据库对象

导入特定的数据库对象使用 INCLUDE 参数，该例子中我们从备份文件中恢复 SCOTT 用户的所有表和触发器对象，而对于已经存在的表则重建后再加载数据。

例子 10-23　使用 INCLUDE 参数导入特定的数据库对象。

```
D:\>impdp scott/tiger@orcl dumpfile=pump dir:MYDB TBS USERSANDSYSTEM 01.DAT
nologfile=y include=table,trigger table_exists_action=replace
```

在备份文件 MYDB_TBS_USERSANDSYSTEM_01.DAT 中，备份了两个表空间即 USERS 和 SYSTEM 中的所有数据库对象，而 SCOTT 用户的数据库对象就存放在这两个表空间中。读者在试验时，可以自己先使用 EXPDP 程序做备份文件，然后再使用 IMPDP 导入特定数据库对象，注意使用哪个用户登录就导入哪个用户的数据库对象。如下使用 LINZI 用户登录（笔者自建的用户）。

例子 10-24　导入特定用户 LINZI 的数据库对象。

```
D:\>impdp linzi/linzi@orcl dumpfile=pump_dir:MYDB_TBS_USERSANDSYSTEM_01.DAT
nologfile=y include=table,trigger table_exists_action=replace
………..
. . 导入了 "LINZI"."CREATE$JAVA$LOB$TABLE"          5.835 KB       1 行
. . 导入了 "LINZI"."DEPT"                           5.656 KB       4 行
. . 导入了 "LINZI"."DEPT_TEST"                      5.789 KB      10 行
. . 导入了 "LINZI"."EMP"                            9.523 KB      56 行
. . 导入了 "LINZI"."EMP_TEST3"                      6.367 KB       4 行
. . 导入了 "LINZI"."ROOMS"                          6.062 KB       8 行
. . 导入了 "LINZI"."SALGRADE"                       5.585 KB       5 行
. . 导入了 "LINZI"."USER MODIFY TABLE"              7.390 KB      68 行
. . 导入了 "LINZI"."BACKUP DELETE EMP TABLE"            0 KB       0 行
. . 导入了 "LINZI"."BONUS"                               0 KB       0 行
. . 导入了 "LINZI"."EMP TEST2"                           0 KB       0 行
……
```

上例只给出了部分输出结果，其他和上例相同。通过这个例子是要告诉读者使用 INCLUDE 过滤了要导入的对象，而登录用户决定了要恢复哪个用户的数据库对象。

10.5　使用数据泵迁移表空间

Oracle 提供了可迁移表空间的新特性，它使得数据库之间移动数据既快速又简单，尤其对于移动大对象数据，使用迁移表空间特性需要将属于源数据库的两个文件，一个是要迁移的表空间的所有数据文件，一个是使用 EXPDP 程序导出的表空间的元数据。将这个两类文件拷贝到目标数据库上，再使用 IMPDP 程序执行导入迁移表空间。

在迁移表空间时需要几个必备的步骤。

- 确定要迁移的表空间，并验证它是否与其他表空间中的对象有依赖关系。
- 导出迁移表空间的元数据，使用 EXPDP 程序生成一个.DMP 表空间元数据备份文件。

- 把备份文件的元数据文件和迁移表空间中的数据文件拷贝到目录数据库。
- 在目标数据库上执行迁移表空间。

下面详细介绍这四个步骤的实现。

01 选择要迁移的表空间。

要迁移的表空间必须满足是自包含的，即该表空间中的对象不能与其他表空间中的对象有依赖关系，所以需要实现验证。Oracle 提供了一个方法，该方法在 DBMS_TTS 程序包中，该方法是 TRANSPORT_SET_CHECK(tbs_name,boolean)。验证方法如下例所示。

例子 10-25 验证表空间是否是自包含。

```
SQL> execute sys.dbms tts.transport set check('users',true);

PL/SQL 过程已成功完成。
```

在上例中，我们使用 EXECUTE 执行过程来验证要迁移的表空间 USERS 是否是自包含的。过程 TRANSPORT_SET_CHECK 没有返回错误消息，说明迁移表空间 USERS 是自包含的，可以作为迁移表空间使用。

02 导出要迁移表空间的元数据。

在将表空间迁移到目标数据库之前，必须使用 EXPDP 导出程序创建迁移表空间的元数据集。这个操作之前必须将要迁移的表空间置为只读状态，如下例所示。

例子 10-26 将表空间置于只读状态。

```
SQL> alter tablespace users read only ;

表空间已更改。
```

要迁移的表空间被置于只读状态后，就可以使用数据泵导出程序 EXPDP 为表空间 USERS 创建元数据备份文件，如下例所示。

例子 10-27 种导出迁移表空间的目录元数据。

```
D:\>expdp system/oracle@orcl dumpfile=pump_dir:transport_users_tbs.dmp
transport_tablespaces=users
Export: Release 11.1.0.6.0 - Production on 星期六, 29 8 月, 2009 11:50:31
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
启动 "SYSTEM"."SYS_EXPORT_TRANSPORTABLE_01":  system/********@orcl
dumpfile=pump_dir:transport_users_tbs.dmp transport_tablespaces=users
处理对象类型 TRANSPORTABLE_EXPORT/PLUGTS_BLK
............
处理对象类型 TRANSPORTABLE_EXPORT/POST_INSTANCE/PLUGTS_BLK
已成功加载/卸载了主表 "SYSTEM"."SYS_EXPORT_TRANSPORTABLE_01"
******************************************************************************
SYSTEM.SYS_EXPORT_TRANSPORTABLE_01 的转储文件集为:
  F:\PUMP\TRANSPORT_USERS_TBS.DMP
作业 "SYSTEM"."SYS_EXPORT_TRANSPORTABLE_01" 已经完成, 但是有 1 个错误 (于
 11:50:47 完成)
```

我们把表空间 USERS 的元数据备份到目录对象 PUMP_DIR 指定的操作系统目录下，因为导出元数据只是导出表空间中数据库对象的定义而不是数据行，所以这个导出过程很快。本例中只导出表空间 USERS 的元数据定义，所以文件很小只有大约 700K。

03 将备份文件和迁移表空间中的数据文件拷贝到目录数据库。

在创建了表空间元数据备份文件后，需要把要迁移的表空间中的所有数据文件和刚才创建的表空间元数据备份文件拷贝到目标数据库的一个可访问目录下。此时可以使用任意的拷贝方式，使用网络传输或者刻成光盘等等。

04 在目标库上导入可迁移表空间。

在笔者笔记本上，将表空间的元数据备份文件和表空间中的所有数据文件都拷贝到目标数据库所在计算机的一个磁盘上，保存目录为目录为 F:\PUMP。下面我们就可以运行 IMPDP 程序在目标数据库中导入源表空间的元数据，目标数据库将使用源表空间中拷贝过来的所有数据文件作为迁移表空间的数据文件，如下例所示。

例子 10-28 在目标数据库中导入迁移表空间。

```
D:\>impdp system/oracle@orcl dumpfile=transport_users_tbs.dmp
transport_datafiles='users01.dbf' directory=pump_dir
Import: Release 11.1.0.6.0 - Production on 星期六, 29 8月, 2009 15:41:27
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
已成功加载/卸载了主表 "SYSTEM"."SYS_IMPORT_TRANSPORTABLE_01"
启动 "SYSTEM"."SYS_IMPORT_TRANSPORTABLE_01":  system/********@orcl
dumpfile=transport_users_tbs.dmp transport_datafiles=
'users01.dbf' directory=pump_dir
处理对象类型 TRANSPORTABLE_EXPORT/CONSTRAINT/CONSTRAINT
………..
作业 "SYSTEM"."SYS_IMPORT_TRANSPORTABLE_01" 已经完成(于 15:41:29 完成)
```

在目标数据库中迁移表空间时，使用 IMPDP 从备份数据中导出迁移表空间的元数据，并创建各种数据库对象，如触发器、表和约束等，但是没有导入任何数据，因为数据已经在拷贝的数据文件中了。TRANSPORT_DATAFILES 参数说明了需要的参数，而迁移的表空间已经在目标数据库上了，至此成功迁移了表空间。

10.6 本章小结

数据泵技术是 Oracle 对于 EXP、IMP 备份与恢复工具的改进，可以理解为是 EXP、IMP 工具的升级版备份与恢复工具，它实现了数据库的逻辑备份。而且使用数据泵可以迁移表空间，这样可以极大提高数据迁移的速度和效率。使用数据泵技术有许多优点，这些优点正是对 EXP\IMP 工具的改进，如多线程作业速度更快、对失败的作业具有记忆功能、可以重启失败的作业、在备份过程中实现交互功能、基于数据库服务器独立于客户机、支持网络操作、可以在两个联网的数据库服务器之间导入和导出数据，以及更加细粒度地控制导入导出的数据库对象。

第 11 章

用户管理的备份与恢复

数据库备份与恢复是 DBA 日常工作的重要部分。本章我们讲解两种用户管理的备份方法，以及相应的数据恢复方法。在最后给出几个典型的用户管理的数据文件恢复案例，通过案例分析使读者对数据恢复有一个清晰地思路并掌握切实可行数据恢复方法。

11.1　用户管理的脱机备份方法

用户管理的脱机备份是指先关闭数据库，而后使用操作系统工具拷贝数据库文件，即数据文件、控制文件和重做日志文件。这种备份总是一致的数据库备份，因为此时用户无法访问数据库，没有数据的变化，备份后的数据库和当前的数据库中的数据是一致的。用户管理的脱机备份遵循以下步骤：

- 首先确认数据库文件所在的操作系统目录，并记录下这些文件的目录信息。
- 关闭数据库，此时不要使用 SHUTDOWN ABORT 关闭数据库。
- 拷贝数据库文件到指定的备份目录中。
- 重启启动数据库，完成备份。

下面我们演示用户管理的脱机备份的具体过程，读者只要按照步骤操作，很容易完成一次脱机备份。

01 查看数据库文件所在的目录，并记录下目录信息。这些数据库文件包括数据文件，控制文件和重做日志文件，如下例所示。

例子 11-1 查看数据文件的存储目录。

```
SQL> col file_name for a55
SQL> select file name, tablespace name
  2* from dba data files

FILE NAME                                         TABLESPACE NAME
--------------------------------------------- ------------------------
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF     USERS
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF    SYSAUX
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF   UNDOTBS1
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF     SYSTEM
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF   EXAMPLE
```

我们看到所有的数据文件都位于一个目录下，在 Oracle 11g 中多了辅助表空间 SYSAUX 表空间中的数据，如果读者安装了 Oracle9i 数据库不会看到该表空间。查看控制文件的存储目录如下例所示。

在笔者的电脑上控制文件位于同一个目录下，而且该目录是 Oracle 默认的控制文件目录，在实际中这样做是很不安全的，需要实现控制文件在不同磁盘的冗余分布。

例子 11-2　查看控制文件的存储目录。

```
SQL> select name from v$controlfile;

NAME
---------------------------------------------------------------------------
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL01.CTL
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL02.CTL
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL03.CTL
```

显然重做日志的也存放在同一个目录下，和控制文件一样这样的做法也是不安全的，在生产数据库中需要把设置多个重做日志组，并且每个日志组有多个重做日志成员，且这些日志成员分布在不同的磁盘上，以提高系统的可靠性。查看重做日志文件的存储目录如下例所示。

例子 11-3　查看重做日志文件的存储目录。

```
SQL> select member
  2  from v$logfile;

MEMBER
---------------------------------------------------------------------------
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO03.LOG
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO02.LOG
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG
```

我们已经知道所有的数据库文件的存储目录，就可以关闭数据库了。

02　关闭 Oracle 数据库，如下例所示。

例子 11-4　安全地关闭 Oracle 数据库。

```
SQL> conn system/oracle@orcl as sysdba
已连接。
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

如果使用 SHUTDOWN ABORT 有可能造成数据丢失，所以为了快速关闭数据库，最好使用 SHUTDOWN IMMEDIATE 安全地关闭数据库。接下来，就是拷贝数据文件了。

03　拷贝数据文件到备份目录。

因为笔者的笔记本上所有的数据库文件都保存在同一目录下，使用一行 host copy 指令就可以完成数据文件的拷贝，如下例所示。

例子 11-5 备份数据文件。

```
    SQL> host copy F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\*.* f:\user  offline
backup
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL01.CTL
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL02.CTL
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\CONTROL03.CTL
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO02.LOG
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO03.LOG
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\TEMP01.DBF
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
    F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
    已复制         12 个文件。
```

注意此时是在数据库关闭状态下执行的，使用 host 指令说明后面的指令是操作系统指令，在 UNIX 系统中使用 host cp 指令拷贝文件到指定目录。此时成功完成数据库的用户管理的脱机备份，然后启动数据库。

04 启动数据库（脱机备份完成后），如下例所示。

例子 11-6 启动数据库完成脱机备份。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1120380 bytes
Variable Size             218106804 bytes
Database Buffers          377487360 bytes
Redo Buffers                7125232 bytes
数据库装载完毕。
数据库已经打开。
```



用户管理的脱机备份中，备份的数据文件只能恢复备份时点之前的数据，如果之后的数据库结构有变化，如创建了表空间、增加了数据文件等，控制文件最好需重新备份。总之用户管理的冷备份使得所有的数据库文件定格在一个时刻，而对于这个时刻之后的数据变化是未知的。并且对于 7×12 小时工作的生产数据库而言，脱机冷备份往往不可取的，需要采取热备份的方式。

11.2　用户管理的联机备份方法

联机备份是指在数据库不关闭的情况下实现备份，而用户管理更强调在这个过程中用户手工操作的过程，比如设置备份模式、拷贝数据文件、备份重做日志文件和归档日志文件等。本节我们将如何实现用户管理的联机备份，显然联机备份是在运行的数据库上实现备份操作，必须将要备份

的表空间置于备份模式。这种模式的含义是告诉数据库该表空间中的数据文件正在备份，不能对它进行修改操作但是可以读取，也不能再向该表空间写入数据，处于备份模式的表空间中的数据不再变化。

那么在备份期间变化的数据，或要写入备份表空间中的数据该如何处理呢？显然需要这个变化的一个备份，读者应该知道归档重做日志的作用吧，这里就使用归档重做日志记录在联机备份期间变化的数据。所以要使用 Oracle 数据库的联机（热）备份，数据库必须处于归档（ARCHIVELOG）模式，并且要求在备份期间产生归档日志。

1. 用户管理的联机热备份的准备工作

从以上叙述，读者应该理解，在实现用户管理的联机热备份前必须做一些准备工作说明如下。

（1）将数据库设置为归档模式

设置数据库为归档模式，在 Oracle 9i 和 Oracle 11g 中方法是一样的。如果数据库打开，则必须首先安全地关闭数据库，再启动数据库到 MOUNT 状态，如下例所示。

例子 11-7 重启数据库到 MOUNT 状态。

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1120380 bytes
Variable Size             234884020 bytes
Database Buffers          360710124 bytes
Redo Buffers                7125232 bytes
数据库装载完毕。
```

然后使用如下例所示的命令将数据库置于归档模式。

例子 11-8 设置数据库为归档模式。

```
SQL> alter database archivelog;

数据库已更改。
```

此时将数据库改变到 OPEN 状态，即在数据库处于 MOUNT 状态时打开数据库，此时可以使用 ARCHIVE LOG LIST 指令查看数据库的归档信息，如下例所示。

例子 11-9 打开数据库。

```
SQL> alter database open;

数据库已更改。
```

例子 11-10　查看数据库的归档信息。

```
SQL> archive log list
数据库日志模式              存档模式
自动存档                启用
存档终点                USE DB RECOVERY FILE DEST
最早的联机日志序列          184
下一个存档日志序列        186
当前日志序列              186
```

从上例的输出可以看出，当前的数据库日志模式处于归档模式，而存档终点是 USE_DB_RECOVERY_FILE_DEST，该值的含义是使用数据库快闪恢复区的数据库存储目录。

（2）设置归档日志存储参数

将参数 LOG_ARCHIVE_DEST_1 的值设置为存储归档日志的目录。如果使用了快闪恢复区作为归档日志文件的存储目录，也可以不设置该参数。

参数 LOG_ARCHIVE_START 的值必须设置为 TRUE。

2. 用户管理的数据文件联机热备份过程

此时用户选择备份的数据文件，可以是整个数据库，也可以是某个数据文件。如果只备份某个表空间的数据文件，可以首先将该表空间置于备份模式；如果要备份整个数据库则需要将所有数据文件所在的表空间置于备份模式。下面我们演示如何联机手工备份一个数据文件。首先需要确定数据文件的操作系统存储目录，以及数据文件对应的表空间，目的是把这些表空间设置为备份模式，如下例所示。

例子 11-11　查找数据文件以及对应的表空间。

```
SQL> col file_name for a55
SQL> select file_name ,tablespace_name
  2* from dba_data_files

FILE_NAME                                              TABLESPACE_NAME
------------------------------- ----------------------------------
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF       USERS
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF      SYSAUX
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF     UNDOTBS1
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF      SYSTEM
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\EXAMPLE01.DBF     EXAMPLE
```

此时，检查数据文件目录目的是拷贝这些文件，而查看表空间的目的是设置这些表空间为备份模式。下面需要将所有表与数据文件相关的表空间置于备份模式，在 Oracle 9i 或以上版本中都可以使用 ALTER TABLESAPCE tbs_name BEGIN BACKUP，将该表空间置于备份模式，如下例所示。

例子 11-12　将表空间设置为备份模式。

```
SQL> alter tablespace users begin backup ;

表空间已更改。
```

此时表空间就置于备份模式，为了拷贝所有的数据文件需要使用上述命令将所有的数据文件

相关的表空间设置为备份模式。在 Oracle10g 及以上版本中，可以使用一条指令将整个数据库置于备份模式，如下例所示。

例子 11-13　将整个数据库置于备份模式。

```
SQL> alter database begin backup ;

数据库已更改。
```

此时，整个数据库的所有表空间都置于备份模式。我们现在可以拷贝数据文件到备份目录了，如下例所示。

例子 11-14　拷贝数据文件到备份目录。

```
SQL> host copy F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
    f:\onlinebackup
已复制         1 个文件。
```

只要重复使用 host copy 指令将需要的数据文件拷贝到备份目录即可。最后需要结束数据库或表空间的备份模式。如将某个表空间退出备份模式，如下例所示。

例子 11-15　把表空间退出备份模式。

```
SQL> alter tablespace users end backup;

表空间已更改。
```

如果在备份数据库时，将整个数据库置于备份模式，则需要使用如下方式将数据库退出备份模式。

例子 11-16　将数据库退出备份模式。

```
SQL> alter database begin backup ;

数据库已更改。
```

3．备份归档日志文件

可以使用如下方式查看当前数据库的归档日志存储目录，从而拷贝在备份过程中产生的归档日志数据。

例子 11-17　查看归档日志位置。

```
SQL> col name for a2
SQL> col name for a12
SQL> col value for a50
SQL> select name,value from v$parameter where name in
  2* ('log_archive_dest','log_archive_dest_1','db_recovery_file_dest')

NAME                      VALUE
------------------------- -----------------------------------------------
log_archive_dest
log_archive_dest_1
db_recovery_file_dest     F:\oracle\product\10.2.0/flash_recovery_area
```

然后到归档日志目录下备份在联机备份过程中（开始备份模式和退出备份模式之间）产生的归档重做日志。

4．备份控制文件

在备份数据文件完成后应该对控制文件做备份，因为在备份数据文件后，整个数据库结构可能发生变化，如新建了数据表空间、当前数据表空间增加了数据文件等等。这些都会记录在控制文件中，而备份的数据文件信息记录在当前的控制文件中，如果在将来需要做数据库的介质恢复，就使用我们备份的数据文件，也需要使用此时备份的控制文件。备份控制文件的方式如下例所示。

例子 11-18 备份控制文件。

```
SQL> alter database backup controlfile to 'f:\onlinebackup\mycontrolfile.ctl';

数据库已更改。
```

下面我们通过操作系统指令查看是否在指定目录 f:\onlinebackup 下生成了控制文件的备份文件，如下例所示。

例子 11-19 查看是否成功创建备份的控制文件。

```
F:\onlinebackup>dir
 驱动器 F 中的卷是 Oracle11g
 卷的序列号是 000B-1DED
F:\onlinebackup 的目录

009-08-30  19:29    <DIR>          .
009-08-30  19:29    <DIR>          ..
009-08-30  18:40         7,061,504 CONTROL01.CTL
………..
009-08-30  19:29         7,061,504 MYCONTROLFILE.CTL
…………
009-08-30  18:40       104,865,792 USERS01.DBF
              12 个文件  2,055,980,544 字节
               2 个目录 70,464,360,448 可用字节
```

从输出看出在操作系统的 F:\onlinebackup 目录下成功备份了控制文件，文件名为 MYCONTROLFILE.CTL。

11.3　备份只读表空间

只读表空间是数据不会发生变化的表空间。在备份只读表空间时，只需要使用操作系统指令拷贝只读表空间对应的数据文件。我们先查看当前的只读表空间信息，如下面例子所示。

例子 11-20 查看的当前的只读表空间和相应的数据文件。

```
SQL> COL FILE_NAME FOR A20
SQL>select a.tablespace_name,b.file_name,a.status
  2  from dba tablespaces a, dba data files b
  3  where a.tablespace_name=b.tablespace_name
```

```
  4* and a.status='READ ONLY'

TABLESPACE_NAME                FILE_NAME          STATUS
------------------------------ -------------------- ---------
INDEX_TBS                     D:\INDEX.DBF         READ ONLY
```

例子 11-21 备份只读表空间的数据文件。

```
SQL> host copy d:\index.dbf e:\indexbackup.dbf
已复制          1 个文件。
```



在恢复只读表空间时，将表空间 OFFLINE，再复原相应的数据文件，再将表空间 ONLINE。如果在备份只读表空间后，该表空间设置为 READ/WRITE 模式，备份的数据文件依然可用，不过此时的复原的数据文件需要恢复（RECOVER）。

11.4　使用...END BACKUP指令恢复表空间备份异常

在使用 ALTER TABLESPACE ...END BACKUP 等指令结束表空间的备份时，会发生一些异常需要处理，如在结束备份时，数据库异常关闭。下面我们详细介绍如何处理这个异常。

11.4.1　使用...END BACKUP 指令恢复表空间备份异常

如果数据库已经异常关闭，而此时的数据文件依然处于备份模式，该数据文件不会被打开，此时需要关闭该数据文件的备份模式，或者对该数据文件实现实例恢复，如下面例子所示。

例子 11-22 将表空间 INDEX_TBS 设置为备份模式。

```
SQL> alter tablespace index_tbs begin backup;

表空间已更改。
```

例子 11-23 异常关闭数据库并尝试重启数据库。

```
SQL> shutdown abort
ORACLE 例程已经关闭。
SQL>
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             234884020 bytes
Database Buffers          360710144 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01113: 文件 5 需要介质恢复
ORA-01110: 数据文件 5: 'D:\INDEX.DBF'
```



此时的数据文件 INDEX.DBF 仍然处于备份模式，所以提示需要介质恢复。

例子 11-24 关闭数据库并启动到 MOUNT 状态。

```
SQL> shutdown immediate
ORA-01109: 数据库未打开


已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount;
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             234884020 bytes
Database Buffers          360710144 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
SQL>
```

例子 11-25 结束备份模式。

```
SQL> alter database end backup;

数据库已更改。
```

例子 11-26 打开数据库。

```
SQL> alter database open;

数据库已更改。
```



在表空间处于备份模式时，如果数据库异常关闭此时的数据不会正常启动，必须在 MOUNT 模式结束表空间的备份模式。

11.4.2 使用 RECOVER DATAFILE 恢复表空间备份期间实例异常

同样我们也可以使用 RECOVER DATAFILE 指令处理表空间备份期间的实例异常，如实例异常关闭等。我们通过一个具体的示例演示异常处理过程。首先将表空间 USERS 设置为备份模式，如下例所示。

例子 11-27 将表空间 USERS 设置为备份模式。

```
SQL> alter tablespace users begin backup;

表空间已更改。
```

接着，验证表空间 USERS 的是否处于备份模式，如下例所示。

例子 11-28 验证表空间 USERS 的是否处于备份模式。

```
SQL> select *
  2  from v$backup;

     FILE# STATUS             CHANGE# TIME
---------- ------------------ ---------- --------------
         1 NOT ACTIVE            1878998 23-6 月 -10
         2 NOT ACTIVE            1878998 23-6 月 -10
         3 NOT ACTIVE            1878998 23-6 月 -10
         4 ACTIVE               1923147 23-6 月 -10
         5 NOT ACTIVE            1882508 23-6 月 -10
         6 NOT ACTIVE            1878998 23-6 月 -10

已选择 6 行。
```

由于此时的数据表空间对应唯一数据文件号为 4，所以我们将表空间设置为备份模式后，此时的 v$backup 视图显示数据文件 4 处于 ACTIVE 状态，即备份模式。接下来异常关闭数据库，模拟实例失效，如下例所示。

例子 11-29 异常关闭数据库，模拟实例失败。

```
SQL> shutdown abort
ORACLE 例程已经关闭。
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             239078324 bytes
Database Buffers          356515840 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01113: 文件 4 需要介质恢复
ORA-01110: 数据文件 4: 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF'
```

上例输出提示，数据文件 4 需要介质恢复，我们使用如下例子完成恢复。

例子 11-30 介质恢复。

```
SQL> recover datafile 4;
完成介质恢复。
SQL> alter database open;

数据库已更改。
```

完成介质恢复后，我们打开了数据库。接下来验证此时的数据文件 4 是否还处于备份模式，如下例所示。

例子 11-31　验证表空间是否还处于备份模式。

```
SQL> select *
  2  from v$backup;

     FILE# STATUS             CHANGE# TIME
---------- ------------------ ---------- --------------
         1 NOT ACTIVE            1878998 23-6月 -10
         2 NOT ACTIVE            1878998 23-6月 -10
         3 NOT ACTIVE            1878998 23-6月 -10
         4 NOT ACTIVE            1923147 23-6月 -10
         5 NOT ACTIVE            1882508 23-6月 -10
         6 NOT ACTIVE            1878998 23-6月 -10

已选择 6 行。
```

显然，此时的数据文件 4 自动结束备份模式，所以在我们使用 RECOVER DATAFILE 恢复没有结束备份模式的表空间对应的数据文件后，该表空间自动结束备份模式。

11.5　备份控制文件

我们可以使用 SQL 的 trace 指令备份控制文件，因为该指令可以获得创建控制文件的基本信息，可以顺利重建一个丢失的控制文件。但是这样的控制文件也有不好的方面，如它不包含归档日志历史、备份集信息以及使用 RMAN 的映像备份信息，而二进制的备份控制文件则都包含这些信息，所以在数据库结构发生变化后，就需要用户管理的备份措施，如下例所示。

例子 11-32　备份二进制控制文件。

```
SQL> alter database backup controlfile to 'd:/ctlbackup.bak' reuse;

数据库已更改。
```



REUSE 指令使得新创建的控制文件覆盖当前的控制文件。

11.6　备份控制文件到TRACE文件

将创建控制文件的 SQL 语句写入 trace 文件，而不是产生一个二进制文件。这些语句包括打开数据库到 MOUNT 状态、重建控制文件、恢复和打开数据库。可以在数据库打开或 MOUNT 状态备份控制文件到 TRACE 文件。如下例所示。

例子 11-33　备份控制文件到 TRACE 文件。

```
SQL> alter database backup controlfile to trace;

数据库已更改。
```

```
如下是 TRACE 文件中的记录的创建控制文件的信息。
-- The following commands will create a new control file and use it
-- to open the database.
-- Data used by Recovery Manager will be lost.
-- The contents of online logs will be lost and all backups will
-- be invalidated. Use this only if online logs are damaged.
-- After mounting the created controlfile, the following SQL
-- statement will place the database in the appropriate
-- protection mode:
--  ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "LSZPRI" RESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO01.LOG'  SIZE 50M,
  GROUP 2 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO02.LOG'  SIZE 50M,
  GROUP 3 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\REDO03.LOG'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF',
  'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF',
  'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF',
  'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF',
  'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\OVRFLW.DBF'
CHARACTER SET ZHS16GBK
;
```

重建控制文件时，需要注意以下几点：

① 对于表空间 ONLINE 而状态为 OFFLINE 的数据文件，在创建控制文件的脚本后添加如下指令：

```
alter database datafile 'd:/test.dbf' offline;
recover database
alter system archive log all
alter database open
```

② 对于只读表空间，此时的备份的控制文件的 SQL 语句不包含只读表空间和正常 OFFLINE 的表空间，对于这样的数据文件 Oracle 会标记为 MISSINGxxxx。因为这些文件不需要恢复，所以只需要重命名丢失的文件，如下例所示。

```
alter database rename file 'missing0001' to 'd:/testb.dbf';
```

11.7　用户管理的全库备份

在归档模式和非归档模式下，都可以进行用户管理的全库备份。如果是非归档模式下，则需

要一致的全库备份，一致的含义是备份的数据库和当前的数据库没有数据差异，二者内容是一样的，所以此时需要先关闭数据库。一致的数据库备份复原时是不需要恢复的，如果一致备份的数据库处于归档模式，则可以将数据库恢复到一个当前的时间点。

如果是归档模式下，则不关心是否是一致性备份，因为有归档文件和重做日志的存在，即使是不一致的数据库备份也可以恢复必要的数据。如果此时是一致性备份，则恢复时可以恢复到从备份到当前之间的任意时间点，因为是归档模式所以有归档文件和重做日志文件可用。

归档模式下的一致备份步骤如下：

01 查找当前的所有数据文件位置，如下例所示。

```
SQL> select file_name
  2  from dba_data_files;

FILE_NAME
-------------------------------------------------------
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSAUX01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\UNDOTBS01.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\SYSTEM01.DBF
D:\INDEX.DBF
E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\OVRFLW.DBF

已选择 6 行。
```



最好将控制文件参数文件以及网络文件一并备份，将来如果有控制文件或网络文件的变化则及时作备份。

02 关闭数据库，如下例所示。

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

03 使用系统指令拷贝数据文件，如下例所示。

```
SQL>host copy E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\*.* f:\backup
SQL>host copy D:\INDEX.DBF f:\backup
```

数据库处于归档模式下，如果由于某种原因数据文件损坏或者丢失，则需要从备份的数据文件中复原丢失的数据文件，然后使用 RECOVER 恢复该数据文件。

下面是一致备份后的恢复实例，前提是数据库处于归档模式。

首先我们在 USERS 表空间中创建一个表 TEST1，如下例所示。

例子 11-34 创建表 TEST1。

```
SQL> create table test1
  2  as
  3  select *
  4  from scott.dept;
```

```
表已创建。
```

接着验证表 TEST1 的内容，如下例所示。

例子 11-35 验证表 TEST1 的内容。

```
SQL> select *
  2  from test1;

   DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

然后，我们查看表 TEST1 对应的表空间信息，如下例所示。

例子 11-36 查看表 TEST1 所在的表空间。

```
SQL> select tablespace_name
  2  from user_tables
  3  where table_name='TEST1';

TABLESPACE_NAME
------------------------------
SYSTEM
```

此时的表 TEST1 存储在 SYSTEM 表空间中，显然这样的存储是不合适的，SYSTEM 表空间应该只存储系统相关的数据库对象。下面我们迁移表 TEST1 到用户表空间 USERS，如下例所示。

例子 11-37 将表 TEST1 迁移到 USERS 表空间并验证。

```
SQL> ALTER TABLE TEST1 MOVE TABLESPACE USERS;

表已更改。
```

成功迁移表 TEST1 到新的表空间 USERS 后，验证一下迁移结果，如下面例子所示。

例子 11-38 查询表 TEST1 的表空间。

```
SQL> select tablespace_name
  2  from user_tables
  3  where table_name='TEST1';

TABLESPACE_NAME
------------------------------
USERS
```

例子 11-39 关闭数据库删除 USERS 表空间对应的数据文件。

```
SQL> shutdown immeidate
SP2-0717: 非法的 SHUTDOWN 选项
SQL> shutdown immediate
```

```
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> select file_name
  2  from dba data files;
select file_name
*
第 1 行出现错误:
ORA-01034: ORACLE not available
```

例子 11-40 重新启动数据库。

```
SQL> startup
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1150380 bytes
Variable Size             251661236 bytes
Database Buffers          343932928 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01157: 无法标识/锁定数据文件 4 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 4: 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF'
```



此时的数据文件 USERS01.DBF 丢失，所以数据库无法正常打开，此时就需要使用备份的数据文件复原到目录 E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\下，我们使用 host copy 操作系统指令将备份恢复到指定目录。

此时，我们不尝试介质恢复，而是尝试打开数据库看提示什么，如下例所示。

例子 11-41 复原数据文件后直接打开数据库。

```
SQL> alter database open;
alter database open
*
第 1 行出现错误:
ORA-01113: 文件 4 需要介质恢复
ORA-01110: 数据文件 4: 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\LSZPRI\USERS01.DBF'
此时提示数据库无法打开，需要介质恢复
```



因为数据库处于归档模式，所以自备份以来，数据库已经发生了变化，为了数据库的一致性，需要将备份的归档数据应用到当前的数据文件中。下面我们实现介质恢复。

例子 11-42 介质恢复数据文件 USERS01.DBF。

```
SQL> recover datafile 4;
完成介质恢复。
```

例子 11-43 打开数据库。

```
SQL> alter database open;

数据库已更改。
```



在此时，我们已经将归档数据写入了复原的数据文件。在备份时表空间 USERS 中没有表 TEST1，而在备份之后我们在表空间 USERS 创建了表 TEST1，此时我们实现了介质恢复，所以表 TEST1 应该是成功恢复的，如下所示测试表 TEST1 是否恢复。

例子 11-44 测试表 TEST1 是否恢复。

```
SQL> select tablespace_name,table_name
  2  from user_tables
  3  where table_name='TEST1';

TABLESPACE_NAME                  TABLE_NAME
------------------------------ ------------------KEJI-----
USERS                            TEST1

可见恢复成功。
```

如果数据库处于非归档模式，则一致的数据库备份方式和上述相同，恢复同样需要介质恢复，此时可能会发生数据丢失。

11.8　从用户管理的脱机（冷）备份中手工恢复

使用用户管理的脱机备份作为恢复数据源时，需要考虑数据库的归档模式。如果数据库处于归档模式，需要使用数据恢复将备份后的所有变化了的数据重写进数据文件中。如果数据库处于非归档模式，则只需要从最近的脱机备份中拷贝数据库的数据文件、联机重做日志文件和控制文件。下面我们分两种情况详细介绍从用户管理的脱机冷备份中手工恢复的方法。

在恢复的数据库不处于归档模式。这种情况下，首先保证数据库处于关闭状态，然后将脱机备份文件拷贝到当前数据库的相应目录下，然后重启数据库。具体步骤如下。

- 关闭数据库，此时可以使用 SHUTDOWN ABORT 来关闭数据库。
- 从最近的脱机备份中拷贝数据库文件，包括数据文件、日志文件和控制文件。此时需要用户事先知道当前数据库的这些文件的位置。
- 重启数据，使用 STARTUP 指令。

上述的恢复方法，可以恢复一个数据文件，或整个数据库。当数据库处于归档模式时，就有所不同了，但它和从热备份执行恢复一样，在重新启动数据库后需要 recover 数据库重做日志中的已提交数据 redo 到数据文件中。

11.9　从联机备份中手工恢复（ARCHIVELOG模式）

当数据库处于归档模式时，可以通过联机备份的数据实现数据恢复，因为联机备份时，处于备份状态的表空间或整个数据库是无法写入数据，虽然可以访问和使用 DML 操作，但是更新的数据只保留在重做日志文件中。所以采用联机备份实现恢复时，需要 RECOVER 数据，即将用户提交的记录在重做日志中的数据重新写到数据文件中。

11.9.1　恢复数据文件

下面通过一个具体例子说明在当前数据库处于归档模式下时，如何实现数据恢复。以恢复一个数据文件为例，假设表空间 USERS 中的数据文件 USERS01.DBF 损坏，或者其中的部分重要的表被删除。具体步骤说明如下。

01 将数据文件 USERS01.DBF 置于脱机状态，之前需要使用数据字典 DBA_DATA_FILES 来查看表空间 USERS 中的数据文件在操作系统上的目录。先将设置数据文件为脱机状态，如下例所示。

例子 11-45　将数据文件设置为脱机状态。

```
SQL> alter database datafile 'F:\ORACLE\PRODUCT\10.2.0\ORADATA\ ORCL\
USERS01.DBF' offline;

数据库已更改。
```



如果在数据库启动时发生数据文件损坏，此时数据库是无法正常启动的，因为控制文件中记录了该数据文件信息，该文件无法打开，数据库会禁止打开数据库，所以此时同样需要将数据文件置为 OFFLINE 状态，这样就可以正常启动数据库了。

02 下面我们查看数据文件 USERS01.DBF 的状态信息，如下例所示。

例子 11-46　查看脱机数据文件 USERS01.DBF 的状态信息。

```
SQL> select file_name,status,online_status
  2  from dba_data_files
  3  where tablespace_name ='USERS';

FILE_NAME                                                      STATUS         ONLINE_
---------------------------------------- --------------------- ----------
F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF  AVAILABLE     RECOVER
```

输出说明文件名 FILE_NAME 为我们需要恢复的数据文件，而 ONLINE_STATUS 的值为 RECOVER，说明该数据文件需要介质恢复，说明数据文件中的 SCN 与控制文件中的 SCN 不一致，需要使用重做日志文件中的数据恢复用户提交的数据。一旦将处于归档模式的数据库中的数据文件设置为脱机状态，则该数据文件的 ONLINE_STATUS 值自动为 RECOVER，指示需要介质恢复。

03 实现数据文件的介质恢复。

此时首先需要将脱机备份的数据文件拷贝到当前数据库中该文件的目录下，然后实现介质恢复。这里我们使用 RECOVER DATAFILE 指令，如下例所示。

例子 11-47　实现数据文件的介质恢复。

```
SQL> recover datafile 'F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF';
完成介质恢复。
```

04 将数据文件设置为在线状态。

虽然实现了介质恢复，但是此时的数据文件仍然是不可访问的，如果试图访问该数据文件中涉及的表，则提示错误如下例所示。

例子 11-48　访问脱机文件涉及的表数据。

```
SQL> select *
  2  from scott.dept;
from scott.dept
          *
第 2 行出现错误:
ORA-00376: 此时无法读取文件 4
ORA-01110: 数据文件 4: 'F:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF'
```

显然，错误很明显由于数据文件没有在线，使得 Oracle 无法读取该文件，下面通过例子将数据文件 USERS01.DBF 设置会在线状态，此时用户就可以访问该数据文件了，如下例所示。

例子 11-49　将数据文件设置为在线状态。

```
SQL> alter database datafile 'F:\ORACLE\PRODUCT\10.2.0\ ORADATA\ ORCL\
USERS01.DBF' online;

数据库已更改。
```

此时，我们成功地通过脱机备份的数据文件为处于归档模式数据库恢复了数据文件 USERS01.DBF。



如果读者是实验数据库，而且有脱机备份的所有数据文件，可以先关闭数据库，然后删除掉 USERS 表空间中的 USERS01.DBF 文件来模拟数据文件丢失。这种情况下，数据库无法正常启动，但是只要使用将脱机的备份数据文件拷贝到当前 USERS01.DBF 数据文件的目录下，然后使用 RECOVER DATAFILE 指令实现数据文件的介质恢复，就可以正常打开数据库了。

使用脱机备份的数据文件在处于非归档模式的数据库上实现数据恢复时，自然也可以恢复表空间和整个数据库。其基本步骤和恢复数据文件类似，我们在接下来的章节中给出具体步骤和相应的注意事项。

11.9.2　使用联机备份恢复表空间

01 确定与表空间相关联的数据文件、记录文件的存储目录，使用如下指令。

```
SQL> select file_name,tablespace_name
  2  from dba_data_files
  3  where tablespace_name ='TBS_NAME';
```

02 将要恢复的表空间置于脱机状态，这样导致该表空间中的数据文件处于脱机状态。设置方法如下例所示，将表空间 USERS 设置为脱机。

```
SQL> alter tablespace users offline;

表空间已更改。
```

03 将最近备份的该表空间中的所有数据文件拷贝到步骤 1 中记录的目录下，注意将备份文件拷贝到当前数据库的表空间中相应的目录下，如 USERS01.DBF 在目录 1 中，则将备份文件中 USERS01.DBF 也拷贝到目录 1 中。

04 使用 RECOVER TABLESPACE 实现表空间的介质恢复，如下例所示。

```
SQL> recover tablespace users;
```

05 将表空间设置为在线状态，如下例所示。

```
SQL> alter tablespace users online;

表空间已更改。
```

11.9.3　使用脱机备份恢复整个数据库

01 如果数据库是可用的，必须先关闭数据库，可以使用 SHUTDOWN ABORT 命令，毕竟需要恢复数据库了，立即关闭数据库已经无所谓了。

02 将备份的所有数据文件拷贝到当前数据库的数据文件的相应目录下。

03 执行数据库恢复，相继使用如下指令。

```
SQL> STARTUP MOUNT;
SQL>RECOVER DATABASE;
SQL>ALTER DATABASE OPEN;
```

首先将数据库置于 MOUNT 状态，我们知道此时数据库打开了控制文件，但是并不判断数据文件的位置，更不会打开数据文件，所以此时执行 RECOVER DATABASE 实现整个数据库的介质恢复。恢复成功后打开数据库，此时数据库会打开恢复的数据文件，使得整个数据库处于可用状态。

11.10　用户管理的典型恢复示例

本节通过几个具体例子演示如何实现用户管理的数据库恢复，典型的恢复案例是由于磁盘损坏造成数据文件丢失或数据文件损坏而无法使用。我们将介绍归档和非归档模式下的数据文件丢失的恢复，以及使用备份的控制文件恢复数据文件或只读表空间的典型示例。

11.10.1　数据文件丢失（非归档模式下）

数据文件的丢失多数情况下是由于磁盘故障引起的，使得数据文件的部分数据损坏，从而使得整个数据文件无法使用，此时就需要使用备份的数据文件来恢复。下面我们分两种情况讨论数据文件丢失情况下的恢复。

1．将数据文件恢复到非原始路径

存在这样一个情景，即当存放数据文件的磁盘损坏时，由于业务的需要往往来不及替换现有磁盘，而是使用已经挂接的磁盘，这样使得业务中断时间尽可能的小，在业务不繁忙的时候可以通过替换坏盘的方式重新部署数据文件。

我们删除了数据文件 USERS01 模拟数据文件丢失，假设该磁盘损坏而需要及时将数据文件恢复到一个全新的磁盘上。下面演示恢复丢失的数据文件到非原始目录的方法，具体操作步骤如下：

01 首先关闭数据库，在生产数据库中如果数据磁盘损坏数据库也会自己关闭，如下例所示。

```
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
```

02 将备份的数据文件 USERS01.DBF 拷贝到新的磁盘目录下，如下例所示。

```
E:\>copy f:\backup\users01.dbf d:\users\users01.dbf
已复制          1 个文件。
```

这里将 d:\users\作为新的磁盘目录使用。

03 启动数据库到 MOUNT 状态，如下例所示。

```
SQL> startup mount;
ORA-32004: obsolete and/or deprecated parameter(s) specified
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             268438452 bytes
Database Buffers          327155712 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
```



启动数据库到 MOUNT 状态目的是修改控制文件中关于数据文件 USERS01.DBF 的记录，高速数据库以前的数据文件存储已经更改，下例就是如何更改控制文件中数据文件记录。

04 重命名数据文件 USERS01.DBF，如下例所示。

```
SQL> alter database rename file 'E:\oracle\product\10.2.0\oradata\lszpri\
users01.dbf' to
'd:\users\users01.dbf' ;

数据库已更改。
```



如果重做日志文件也需要存储在不同的磁盘目录下，修改方式和修改数据文件的磁盘存储目录相同。

05 实现介质恢复，如下例所示。

```
SQL>RECOVER DATABASE UNTIL CANCEL
……
……
CANCEL;
```

06 打开数据库（RESETLOGS），如下例所示。

```
SQL> alter database open resetlogs;

数据库已更改。
```



这种方式将清楚在线重做日志文件并将日志序列号置为 1，这样数据库将忽略从数据文件备份到故障发生时的所有数据库变化信息。

07 验证恢复结果，如下例所示。

例子 11-50　查询当前的数据文件部署。

```
SQL> COL FILE_NAME FOR A40
SQL>select tablespace_name,file_name
  2  from dba_data_files
  3* where tablespace_name='USERS'

TABLESPACE_NAME              FILE_NAME
---------------------------------------------------------
USERS                      D:\USERS\USERS01.DBF
```

从输出看出数据文件 USERS01.DBF 已经被成功迁移到目录 D:\USERS\下。



其他文件如控制文件等存储路径变化也使用了类似的方式恢复，需要修改参数文件中 CONTROL_FILES 参数的值，说明新的控制文件存储目录。

2．将数据文件恢复到原始路径

如果恢复数据文件到默认原始存储路径，则参照上面的步骤省略去步骤 4，并在步骤 2 将备份的数据文件拷贝到默认存储路径下即可，其他步骤相同。

11.10.2　数据文件丢失（归档模式下）

当前的数据库处于归档模式，但是由于磁盘故障造成数据文件丢失或者损坏，此时需要恢复丢失的数据文件。假设此时数据库被强迫关闭，这个场景是当前数据库处于归档模式，在数据文件受损时数据库关闭，此时如何恢复数据文件。



如果仅仅是磁盘控制故障，数据文件没有损坏，此时可以修复磁盘控制器后，启动数据库即可。

如果是磁盘损坏，并且相关的数据文件也损坏，在我们更换当前磁盘后，继续使用以前的数据分区，即数据文件的默认目录不变，我们还是使用这个目录存储来恢复损坏的数据文件，我们进入如下的恢复过程，具体步骤如下：

01 将数据文件从备份目录拷贝到原始目录。

02 启动数据库到 MOUNT 状态。

03 数据恢复，此时要确保相应的数据文件 ONLINE，并使用如下命令：

```
RECOVER DATAFILE datafile。
```

04 打开数据库。



如果是恢复整个数据库、表空间或者某个数据文件则使用如下指令：

恢复数据库：recover database。

恢复表空间：recover tablesapce users。

恢复数据文件：recover datafile 4。

11.10.3 使用备份的控制文件恢复新添加的数据文件

该恢复案例基于这样一个场景，即数据库处于非归档模式，先备份数据库的控制文件，然后在系统的运行期间创建了新的表空间，此后的某个时间使用备份的控制文件执行数据库恢复。在这个案例中关键是老的控制文件不知道新的数据文件的存在，而 Redo 又需要恢复该表空间中。

下面我们模拟这个恢复过程，具体步骤如下。

01 关闭数据库并冷备份控制文件。

02 重新打开数据库。

03 创建新的表空间，如下例所示。

例子 11-51 创建新的表空间 TEST。

```
SQL> create tablespace test
  2  datafile 'd:\test.dbf' size 100m;

表空间已创建。
```

创建新的表空间后，此时的控制文件内容已经与备份时的控制文件内容不同，即此时的控制文件已经包含了新创建的表空间信息。

在数据库运行过程中由于控制文件丢失或磁盘损坏，所以此时需要使用备份的控制文件来恢复数据库。

04 关闭数据库。其实，如果所有的控制文件都无法使用，此时数据库会自动关闭。

05 将备份的控制文件拷贝到原始控制文件所在目录，如果原始控制文件存储目录更换，则

需要修改参数文件中控制文件的位置信息。此处，我们假设将控制文件恢复到原始目录。

06 尝试启动数据库，会提示如下错误。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                 1250380 bytes
Variable Size            255855540 bytes
Database Buffers          339738624 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
ORA-01122: 数据库文件 1 验证失败
ORA-01110: 数据文件 1: 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF'
ORA-01207: 文件比控制文件更新 - 旧的控制文件
```

此时，数据文件比控制文件更新，造成此时的数据库文件 1 验证失败，下面再使用 ALTER DATABASE USING BAKCUP CONTROLFILE UNTIL CANCEL 指令恢复数据文件，如下例所示。

例子 11-52 恢复数据文件。

```
SQL> recover database using backup controlfile until cancel;
ORA-00279: 更改 772495 (在 06/29/2010 19:27:52 生成) 对于线程 1 是必需的
ORA-00289: 建议:
E:\ORACLE\PRODUCT\10.2.0\FLASH RECOVERY AREA\ORCL\ARCHIVELOG\2010 06 29\O1
_MF_1_
1 %U .ARC
ORA-00280: 更改 772495 (用于线程 1) 在序列 #1 中


指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
E:\oracle\product\10.2.0\oradata\orcl\redo01.log
ORA-00283: 恢复会话因错误而取消
ORA-01244: 未命名的数据文件由介质恢复添加至控制文件
ORA-01110: 数据文件 5: 'D:\TEST.DBF'


ORA-01112: 未启动介质恢复
```

其中，ORA-01244 说明已经将未命名的数据文件添加到控制文件中，这个未命名的数据文件就是 ORA-01110 指定的数据文件。



上例的粗体部分是用户输入的内容。

我们也可以从告警日志文件中获得这个未命名的文件信息，如下代码所示。

```
Media Recovery Log E:\oracle\product\10.2.0\oradata\orcl\redo01.log
File #5 added to control file as 'UNNAMED00005'. Originally created as:
'D:\TEST.DBF'
```

我们通过数据字典视图 v$datafile 来查看当前控制文件知道的数据文件，当然其中包含未知

（UNNAMED）的数据文件，如下例所示。

例子 11-53 查看当前的数据文件。

```
SQL> select file#,name
  2  from v$datafile;

     FILE# NAME
---------- ---------------------------------------------------------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF
         5 E:\ORACLE\PRODUCT\10.2.0\DB_1\DATABASE\UNNAMED00005
```

从输出我们看到数据文件 5 是控制文件未知的数据文件，而从告警日志文件中，我们知道该文件的原始（Original）文件为“D:\TEST.DBF”，所以进入下一步。

07 重命名未知的数据文件。下面我在数据库处于 MOUNT 状态时重命名该未知的数据文件为已知的数据文件“D:\TEST.DBF”，如下例所示。

例子 11-54 重命名未知的数据文件。

```
SQL> alter database rename file
'E:\ORACLE\PRODUCT\10.2.0\DB 1\DATABASE\UNNAMED00006' to 'd:\new tbs.dbf';

数据库已更改。
```

此时，已经成功将未知的数据文件重命名为已知的数据文件“d:\new_tbs.dbf”，此时我们再次查看当前控制文件知道的数据文件，则不会发现未知的数据文件，如下例所示。

例子 11-55 再次查看当前的数据文件。

```
SQL> select file#,name,status
  2  from v$datafile;

     FILE# NAME                                                  STATUS
--------------- ------------------------- ------------------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF    SYSTEM
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF   ONLINE
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF    ONLINE
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF      ONLINE
         5 D:\TEST.DBF                                          RECOVER
```

此时，我们看到的是数据文件 5 为已知的数据文件，该文件就是我们在备份控制文件后创建表空间 TEST 时的数据文件，但是其状态为 RECOVER 说明需要恢复该数据文件。

08 使用 RECOVER DATABASE USING BAKCUP CONTROLFILE 指令恢复数据库，如下例所示。

例子 11-56 使用备份的控制文件恢复数据库。

```
SQL> recover database using backup controlfile until cancel;
ORA-00279: 更改 768455 (在 06/29/2010 19:06:52 生成) 对于线程 1 是必需的
```

```
    ORA-00289: 建议:
    E:\ORACLE\PRODUCT\10.2.0\FLASH_RECOVERY_AREA\ORCL\ARCHIVELOG\2010_06_29\O1
_MF_1_
    1_%U_.ARC
    ORA-00280: 更改 768455 (用于线程 1) 在序列 #1 中


    指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
    E:\oracle\product\10.2.0\oradata\orcl\redo01.log
    已应用的日志。
    完成介质恢复。
```

此时，提示已经成功应用日志，其根源是此时的控制文件知道了数据文件 D:\TEST.DBF 的存在，那么就可以将 Redo 数据写入该文件。

09 使用 RESETLOGS 指令打开数据库，如下例所示。

例子 11-57 打开数据库。

```
SQL> alter database open resetlogs;

数据库已更改。
```

10 再次查看此时数据文件以及相应状态，如下例所示。

例子 11-58 查看数据文件以及相应状态。

```
SQL> col file name for a55
SQL> select file#,name,status
  2  from v$datafile;

     FILE# NAME                                                         STATUS
---------- ------------------------------------------------------------ -------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF           SYSTEM
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF          ONLINE
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF           ONLINE
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF            ONLINE
         5 D:\TEST.DBF                                                  ONLINE
```

从输出知道数据文件 5 已经是 ONLINE 状态了，说明已经成功恢复数据文件 5。

11.10.4 无备份条件下的数据文件重建

数据文件丢失，且数据文件没有备份，如果数据库处于归档模式（从原始文件创建开始就所有的归档数据没有丢失），并且控制文件中记录的丢失的数据文件记录。我们模拟这种故障，并实现数据恢复。

首先，我们创建一个表空间 TEST 和数据文件 TEST.DBF，并在该数据文件中存储一个表，此时的数据库已经处于归档模式。然后我们关闭数据，并删除该数据文件，模拟数据文件丢失。下面是详细的步骤。

01 查看表空间 TEST 和相应的数据文件，方法如下所示。

```
SQL> col file_name for a30
SQL>select tablespace_name,file_name,status
  2  from dba_data_files
  3* where tablespace_name ='TEST'

TABLESPACE NAME          FILE NAME                   STATUS
-------------------- --------------------- -------------------------
TEST                      D:\TEST.DBF                  AVAILABLE
```

02 验证表空间中的表，方法如下所示。

```
SQL> select table name,tablespace name
  2  from user tables
  3  where table name ='TEST EMP';

TABLE NAME                    TABLESPACE NAME
------------------------- ------------------------------
TEST_EMP                      TEST
```

输出说明，表 TEST_EMP 在表空间 TEST 中。

03 主动产生归档，方法如下所示。

```
SQL> alter system switch logfile;

系统已更改。
```

此时，如果读者在闪回恢复区会看到一个新的归档文件。然后关闭数据库，删除数据文件 TEST.DBF，再重启数据库。

04 数据文件丢失后重启数据库，方法如下所示。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                 1250380 bytes
Variable Size            171969460 bytes
Database Buffers         423624704 bytes
Redo Buffers               7135232 bytes
数据库装载完毕。
ORA-01157: 无法标识/锁定数据文件 5 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 5: 'D:\TEST.DBF'
```

此时，提示数据文件无法锁定，我们模拟了一次数据文件损坏事件，但是此时没有数据库备份。下面演示在没有备份的条件下如何，恢复该数据文件。

05 创建新的数据文件替换损坏的数据文件，方法如下所示。

```
SQL> alter database create datafile 'D:\TEST.DBF'
  2  as
  3  'e:\test.dbf';

数据库已更改。
```

上述语句成功在“E: \”目录创建新的数据文件，Oracle 数据库通过控制文件和数据字典知道

原始文件的大小等信息，这样重建的数据文件“E:\TEST.DBF”与原始数据文件具有相同的大小，并且新数据文件替换为原始的数据文件信息记录在控制文件中。



此时数据库处于 MOUNT 状态，因为数据文件丢失或损坏无法打开数据库，但是如果出于业务需要可以在 MOUNT 状态下，先将该数据文件 OFFLINE，

06 实施介质恢复，方法如下所示。

```
SQL> recover datafile 'e:\test.dbf';
完成介质恢复。
```

此时，将归档文件中的相关数据恢复到新的数据文件中。

07 打开数据库，方法如下所示。

```
SQL> alter database open;

数据库已更改。
```

此时，数据库正常打开，而且丢失的数据文件通过归档日志获得恢复。

下面我们验证恢复结果，首先检查数据文件和表空间 TEST 的数据文件是否变化。

08 查看表空间 TEST 的新数据文件，方法如下所示。

```
SQL> select tablespace_name,file_name
  2  from dba_data_files
  3  where tablespace_name='TEST';

TABLESPACE_NAME                FILE_NAME
------------------------------ ------------------------------
TEST                           E:\TEST.DBF
```

此时，表空间 TEST 原始的数据文件 D:\TEST.DBF 被新的数据文件 E:\TEST.DBF 所代替。那么数据是否得以恢复呢，我们查看表 TEST_EMP 是否恢复。

09 验证表 TEST_EMP 是否成功恢复，方法如下所示。

```
SQL> select tablespace name,table name
  2  from user tables
  3  where table name='TEST EMP';

TABLESPACE NAME                  TABLE NAME
-------------------------------- --------------------------
TEST                             TEST_EM
```

从输出结果看出表 TEST_EMP 也成功恢复到表空间 TEST 中，此时我们已经可以确认丢失的数据文件已经得到成功恢复。这里成功的含义是原始数据文件被新的数据文件代替，并且自原始数据文件创建之初的所有归档数据都得以恢复。

11.10.5　恢复 NOLOGGING 的表和索引

在创建表或索引时，使用 NOLOGGING 的含义是不产生 Redo 记录，以这种方式创建的表不

受 Redo 的保护，显然没有 Redo 记录，就更不会记录到归档日志中了，所以即使数据库处于归档模式，也无法通过归档日志恢复这样的数据库对象。

要恢复这样的表或索引只有通过对他们的备份来恢复。下面我们在表空间 TEST 中使用 NOLOGGING 参数创建一个表 TEST_DETP1，做一系列的测试以验证我们上面的判断，如下例所示。

例子 11-59　使用 NOLOGGING 参数创建表。

```
SQL> run
  1  create table test_dept1
  2  tablespace test
  3  nologging
  4  as
  5  select *
  6* from scott.dept

表已创建。
```

以上，我们创建了一个表 TEST_DEPT，且使用了 NOLOGGING 选项，这样该表的创建操作就不会记录在 Redo 中。我们继续查看该表的属性，如下例所示。

例子 11-60　查看表 TEST_DEPT 的表空间以及 NOLOGGING 属性。

```
SQL> select table name,tablespace name,logging
  2  from user tables
  3  where table name='TEST DEPT1';

TABLE NAME                      TABLESPACE NAME                 LOG
------------------------- --------------------------- ----------
TEST_DEPT1                       TEST                            NO
```

从输出看出该表的 LOGGING 列记录为 NO ，说明该表不受 Redo 保护，下面我们测试是否可以通过归档恢复，以及如何通过备份恢复这些表。

我们先强制一次归档，方法如下所示。其实此时表 TEST_DETP1 不受 Redo 保护，所以此时的归档记录也无法保护该表的数据。

例子 11-61　强制当前的数据库归档。

```
SQL> alter system switch logfile;

系统已更改。
```

接着，我们备份该表为以后的恢复做准备。使用 EXP 工具导出表 TEST_DEPT1，如下例所示。

例子 11-62　导出表 TEST_DEPT1。

```
E:\>exp  tables=test dept1 file=tables.dmp

Export: Release 10.2.0.1.0 - Production on 星期六 6月 26 14:10:27 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.
```

```
用户名: sys as sysdba
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的表通过常规路径...
. . 正在导出表                        TEST_DEPT1 导出了          4 行
成功终止导出，没有出现警告。
```

该步骤对表 TEST_DEPT1 做备份，下面我们删除表 TEST_DEPT1，然后通过 RECOVER 指令恢复数据表空间 TEST 中数据文件 E:\TEST.DBF，如下例所示。

例子 11-63 删除表 TEST_DEPT1。

```
SQL> drop table test_dept1 purge;

表已删除。
```

此时表 TEST_DEPT1 从数据库中永久删除，我们下面尝试从归档日志恢复。首先将数据文件 E:\TEST.DBF 的状态变为 OFFLINE，如下例所示。

```
SQL> alter tablespace test offline;

表空间已更改。
```

接着，使用新的数据文件 D:\TEST.DBF 代替原始的数据文件 E:\TEST.DBF，如下例所示。

例子 11-64 使用新的数据文件 D:\TEST.DBF 代替原始的数据文件 E:\TEST.DBF。

```
SQL> alter database create datafile 'e:\test.dbf'
  2  as 'd:\test.dbf';

数据库已更改。
```

此时，新的数据文件是空的，我们可以通过归档日志来恢复自该数据文件创建以来的所有数据，以验证表 TEST_DEPT1 是否可以恢复，如下例所示。

例子 11-65 RECOVER 数据文件 D:\TEST.DBF。

```
SQL> recover datafile 'd:\test.dbf';
…..
已应用的日志。
完成介质恢复。
```

此时，我们使用归档日志恢复了表空间 TEST 创建以来所有被 Redo 保护的数据，而我们知道表 TEST_DEPT1 在创建时使用了 NOLOGGING，因此它不受 Redo 保护，此时不应该被恢复。下面我们从查询表 TEST_DEPT1 中的数据，观察数据库的反应，如下例所示。

例子 11-66 查看表空间 TEST 中表 TEST_DEPT1 是否恢复。

```
SQL> select *
  2  from test_dept1;
```

```
from test_dept1
     *
第 2 行出现错误:
ORA-00942: 表或视图不存在
```

从输出看出，因为数据块是使用 NOLOGGING 选项加载的，所以该数据块无法通过归档日志恢复。此时，只有一个办法就是从备份中恢复，我们已经使用 EXP 工具备份了表 TEST_DEPT1，此时就通过备份恢复该表，如下例所示。

例子 11-67 从 EXP 备份中恢复表 TEST_DEPT1。

```
E:\>imp tables=test_dept1 file=tables.dmp

Import: Release 10.2.0.1.0 - Production on 星期六 6月 26 14:22:38 2010

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

用户名: sys as sysdba
口令:

连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.02.01 创建的导出文件
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
. 正在将 SYS 的对象导入到 SYS
. 正在将 SYS 的对象导入到 SYS
. . 正在导入表                  "TEST_DEPT1"导入了           4 行
成功终止导入, 没有出现警告。
```

上述输出提示表 TEST_DEPT1 成功导入，接着我们验证表是否在表空间 TEST 中以及表的 LOGGING 属性，如下例所示。

例子 11-68 验证表是否在表空间 TEST 中以及表的 LOGGING 属性。

```
SQL> select table_name,tablespace_name,logging
  2  from user_tables
  3  where table_name='TEST_DEPT1';

TABLE_NAME                    TABLESPACE_NAME                 LOG
------------------------ ------------------------------- --------
TEST_DEPT1                     TEST                            NO
```

此时，我们通过 IMP 指令成功地从逻辑备份恢复了表 TEST_DEPT2，该表会默认恢复到原始的表空间 TEST 中，并且 LOGGING 属性不变依然是 NO。

如果此时需要使用 Redo 保护表 TEST_DEPT1，则可以通过如下指令修改表的 LOGGING 属性，如下所示。

例子 11-69 修改表 TEST_DEPT1 的 LOGGING 属性。

```
SQL> alter table test_dept1 logging;

表已更改。
```

再次查询表 TEST_DEPT1 的 LOGGING 属性是否成功修改，如下例所示。

例子 11-70 验证表 TEST_DEPT1 的 LOGGING 修改结果。

```
SQL> select table_name,tablespace_name,logging
  2  from user_tables
  3  where table_name='TEST_DEPT1';

TABLE_NAME                     TABLESPACE_NAME                LOG
------------------------------ ------------------------------ ---------
TEST_DEPT1                         TEST                           YES
```

LOG 属性值为 YES 说明表 TEST_DETP1 已经受到 Redo 的保护。以后该表的数据 DML 操作如 INSERT、UPDATE、DELETE 将记录到 Redo 中。

11.10.6 使用重建的控制文件恢复只读表空间

本节的前提条件是当前的数据库已经存在一个只读表空间，在只读表空间存在的条件下，备份一个控制文件。我们使用 ALTER SYSTEM BACKUP CONTROFIEL TO TRACE 指令备份控制文件的脚本。

由于磁盘损坏或者其他原因造成控制文件丢失，此时需要使用创建控制文件的脚本来重新创建控制文件，而对于只读表空间不会记录在控制文件中，但是数据字典中记录了只读表空间的存在，所以在 RECOVER 数据库时，需要额外的操作使得控制文件知道只读表空间中的数据文件。我们通过如下的具体步骤来演示这个案例。

01 重建控制文件，此时应用备份的控制文件脚本来创建，如下例所示。

例子 11-71 重建控制文件。

```
SQL> startup nomount
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1250380 bytes
Variable Size             209718196 bytes
Database Buffers          385875968 bytes
Redo Buffers                7135232 bytes
SQL> CREATE CONTROLFILE REUSE DATABASE "ORCL" NORESETLOGS  NOARCHIVELOG
  2      MAXLOGFILES 16
  3      MAXLOGMEMBERS 3
  4      MAXDATAFILES 100
  5      MAXINSTANCES 8
  6      MAXLOGHISTORY 292
  7  LOGFILE
  8    GROUP 1 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG'  SIZE 50M,
  9    GROUP 2 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO02.LOG'  SIZE 50M,
 10    GROUP 3 'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO03.LOG'  SIZE 50M
 11  -- STANDBY LOGFILE
 12  DATAFILE
 13    'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF',
```

```
  14   'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF',
  15   'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF',
  16   'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF'
  17  CHARACTER SET ZHS16GBK
  18  ;

控制文件已创建。
```

在控制文件创建成功后，在告警日志文件中会记录如下信息：

```
Dictionary check beginning
Tablespace 'TEMP' #3 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'TEST' #7 found in data dictionary,
but not in the controlfile. Adding to controlfile.
File #5 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00005' in the controlfile.
Dictionary check complete
```

其中表空间 TEMP#3 和 TEST#7 记录在数据字典中，但是没有记录在控制文件中，因为临时文件和只读表空间不需要恢复，所以也就不需要在控制文件中记录，同样数据文件 5 也是如此，它在数据字典中存在但是控制文件中没有，所以此时都将添加到控制文件中，将 File#5 置 OFFLINE 并命名为 MISSING00005。

02 启动数据库，如下例所示。

例子 11-72 启动数据库。

```
SQL> alter database open;

数据库已更改。
```

然后查看当前控制文件中记录的数据文件的状态，如下例所示。

例子 11-73 查看当前数据文件的状态。

```
SQL> col name for a55
SQL>select file#,name,status
  2* from v$datafile

    FILE# NAME                                                    STATUS
---------- ------------------------------------------------------- -------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF      SYSTEM
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF     ONLINE
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF      ONLINE
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF       ONLINE
         5 E:\ORACLE\PRODUCT\10.2.0\DB_1\DATABASE\MISSING00005     OFFLINE
```

此时，存在一个未知的数据文件，它对应于表空间 TEST，其实这个数据文件就是在重新创建控制文件前的只读表空间 TEST 的数据文件，但是由于重建控制文件造成该文件无法识别。

我们继续查询表空间 TEST 对应的数据文件。

例子 11-74 查看表空间 TEST 对应的数据文件。

```
SQL> col file name for a52
SQL> run
  1  select tablespace name,file name,status,online status
  2* from dba data files

TABLESPACE FILE NAME                                        STATUS     ONLINE
---------- ---------------------------------------------- ---------- -------
USERS  E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF    AVAILABLE ONLINE
SYSAUX E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF   AVAILABLE ONLINE
UNDOTBS1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF AVAILABLE ONLINE
SYSTEM E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF   AVAILABLE SYSTEM
TEST E:\ORACLE\PRODUCT\10.2.0\DB_1\DATABASE\MISSING00005  AVAILABLE OFFLINE
```

表空间 TEST 为 READ ONLY 表空间，所以其数据文件没有记录在控制文件中，在使用脚本创建新的控制文件时，该文件没有记录在控制文件中，但是数据字典中有该文件的记录，所以在控制文件中对该文件默认做了未知命名。

03 重命名数据文件。

为了正常使用数据文件，我们必须将表空间 TEST 对应的数据文件修改为 D:\TEST.DBF，如下例所示。

例子 11-75 重命名文件。

```
SQL> alter database rename file 'E:\ORACLE\PRODUCT\10.2.0\ DB_1\DATABASE\MISSING00005'
 to 'd:\test.dbf';

数据库已更改。
```

04 然后，将表空间 TEST 设置为 ONLINE，如下例所示。

例子 11-76 将表空间 TEST 设置为 ONLINE。

```
SQL> alter tablespace test online;

表空间已更改。
```

接着继续查询当前数据文件的状态，如下例所示。

例子 11-77 查询当前数据文件的状态。

```
SQL> select file#,name,status
  2  from v$datafile;

    FILE# NAME                                                       STATUS
---------- -------------------------------------------------------- -------
         1 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSTEM01.DBF       SYSTEM
         2 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\UNDOTBS01.DBF      ONLINE
         3 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\SYSAUX01.DBF       ONLINE
         4 E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\USERS01.DBF        ONLINE
         5 D:\TEST.DBF                                              ONLINE
```

从输出知道只读表空间 TEST 的数据文件已经 ONLINE，可以供用户使用了，此时成功地从重建的控制文件中恢复了只读表空间。

11.11　本章小结

用户管理的备份与恢复是指用户使用操作系统指令来备份数据库文件，是一种数据库的物理备份。用户管理的备份和恢复既可以脱机执行也可以联机执行。

用户管理的脱机备份需要先关闭数据库，而后使用操作系统工具拷贝数据库文件，即数据文件、控制文件和重做日志文件，这种备份总是一致的数据库备份。

联机备份是指在数据库不关闭的情况下实现备份，用户管理强调在这个过程中用户手工操作的过程，比如设置备份模式、拷贝数据文件、备份重做日志文件和归档日志文件等。联机备份要求数据库处于归档模式，而在备份数据库对象如特定表空间时，备份的表空间置于备份模式，处于备份模式的表空间中的数据不再变化。数据文件丢失或损坏后的恢复是十分典型的恢复案例，在本节中，我们给出了归档以及非归档模式下数据文件丢失的恢复过程。

第 12 章

Oracle闪回技术

Oracle 的闪回技术是一种数据恢复技术，具有恢复时间快，不使用备份文件的特点。它使得数据库可以回到过去的某个状态，可以满足用户的逻辑错误的快速恢复。本章我们讲解两种闪回技术，闪回删除和闪回数据库。无论是哪种闪回技术都是传统的基于时间点恢复的技术改进，都可以快速、便捷地恢复用户错误或恢复数据库。

12.1 理解闪回级别

闪回级别也可以理解为闪回粒度，针对闪回删除以及闪回数据库，可以定义两种数据库的闪回级别，即表级闪回以及数据库级闪回。

- 表级闪回：表级闪回可以将表闪回到过去的某个时间点，或恢复到过去的某个 SCN，而闪回删除闪回使用 DROP 指令删除的表。
- 数据库级闪回：数据库级闪回允许将整个数据库恢复到过去的某个时间点。数据库级的恢复在这种情况下使用：当误删除一个用户，或者误截断的一个表时，可以采用数据库级的闪回恢复。

下面将详细介绍闪回删除和闪回数据库技术。

12.2 闪回删除

闪回删除的目的是防止用户错误地删除表、索引等数据库对象。在未使用闪回技术之前，只能使用传统的数据恢复方式从备份中恢复，此时需要备份文件以及归档日志文件。这样的恢复往往涉及哪些不需要恢复的对象，显然这样的恢复不具有针对性，而使用闪回删除就可以直接恢复想要恢复的对象，更高效省时。

12.2.1 闪回删除原理

如果使用 DROP TABLE 指令删除表，该表不会从数据库中立即删除，而是保持原表的位置，但是将删除的表重新命名，并将删除的表信息存储在回收站中，回收站记录了被删除表的新名字和原名字。显然此时被删除的表所占有的空间没有被立即释放，只是数据库可以使用的潜在空间。记

录在回收站中的信息会保留一段时间，直到回收站空间不足或者使用 PURGE 指令删除回收站中的记录。

回收站是一个逻辑结构，不具有物理数据结构，只要删除的表信息记录在回收站中，就可以通过闪回技术恢复删除的表。

如图 12-1 所示，第一步删除表 employees，然后该表对应的物理数据块空间就成为备用的可用空间，该删除记录保存记录在回收站中。回收站为删除的表重新命名并记录该表的原始名，这样用户就很容易查询删除的表以及对应的原始表名。

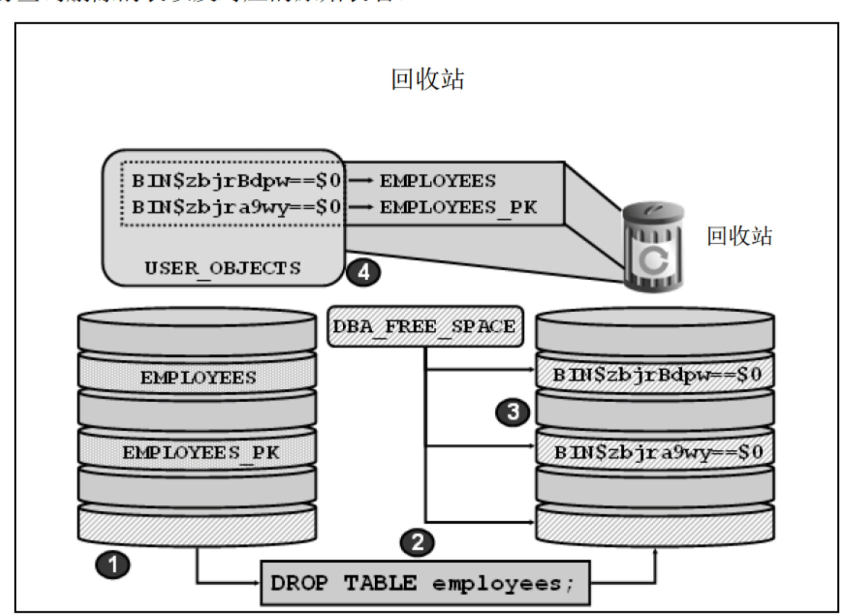


图 12-1　闪回删除原理图

下面我们查看回收站的相关信息，如删除对象、删除时间等，如下例所示。

例子 12-1 查看回收站信息。

```
SQL>show recyclebin;

ORIGINALN   RECYCLEBIN NAME                  OBJECT TYPE DROP TIME
---------   ------------------------------   ----------- ------------------
EMPLOYEES   BIN$xJlIkIFVR/yaulQb8ocxx!==$0   TABLE       2010-05022:22:00:33
```

要启动闪回删除，关键是启动 recyclebin，我们可以通过 alter system set 指令动态设置该参数。默认 Oracle 启动闪回删除，查询参数 RECYCLEBIN 的值如下例所示。

例子 12-2 查询参数 RECYCLEBIN 的值。

```
SQL> show parameter recyclebin;

NAME                            TYPE                  VALUE
------------------------------ -------------------- ----------------
recyclebin                      string                on
```

显然参数 RECYCLEBIN 的值为 ON，所以当前数据库上启动了闪回删除。如果该参数值显示为 OFF，则使用 alter system set 指令启动闪回删除如下例所示。

例子 12-3 启动闪回删除。

```
SQL> alter system set recyclebin=on scope=both;

系统已更改。
```

12.2.2　回收站的使用

在闪回删除原理中，我们已经理解了回收站其实是一个逻辑结构，记录了被删除表的逻辑信息。Oracle 提供了数据字典视图 USER_RECYCLEBIN 和 DBA_RECYCLEBIN 供用户查询数据库中回收站中记录的信息。

我们先看数据字典 DBA_RECYCLEBIN 的结构。

```
SQL> desc dba recyclebin;
 名称                                          是否为空?         类型
 --------------------------------------------- --------- -----------------

 OWNER                                         NOT NULL         VARCHAR2(30)
 OBJECT_NAME                                   NOT NULL         VARCHAR2(30)
 ORIGINAL_NAME                                                  VARCHAR2(32)
 OPERATION                                                      VARCHAR2(9)
 TYPE                                                           VARCHAR2(25)
 TS_NAME                                                        VARCHAR2(30)
 CREATETIME                                                     VARCHAR2(19)
 DROPTIME                                                       VARCHAR2(19)
 DROPSCN                                                        NUMBER
 PARTITION_NAME                                                 VARCHAR2(32)
 CAN_UNDROP                                                     VARCHAR2(3)
 CAN_PURGE                                                      VARCHAR2(3)
 RELATED                                       NOT NULL         NUMBER
 BASE_OBJECT                                   NOT NULL         NUMBER
 PURGE_OBJECT                                  NOT NULL         NUMBER
 SPACE                                                          NUMBER
```

由于在闪回删除中回收站的重要作用，我们详细解释该数据字典的结构，说明如下。

- OWNER:　　表示被删除的表所属用户。
- OBJECT_NAME:　　Oracle 为删除的表的重命名。
- ORIGINAL_NAME:　　被删除表的原始表名。
- OPERATION:　　对表的操作，因为是删除表，所以该列会显示为 DROP。
- TYPE:　　被删除的数据库对象类型，如表或索引等。
- TS_NAME:　　被删除的数据库对象对应的表空间。
- CREATETIME:　　回收站中被删除对象的创建时间。
- DROPTIME:　　删除时间
- CAN_UNDROP:　　记录对象是否可以闪回删除。

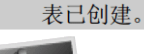
- CAN_PURGE: 该记录是否可以被永久删除。

下面通过一个例子充分理解闪回删除回收站的记录信息。首先创建一个测试表，然后删除该表，查看该表在回收站中的记录，如下面例子所示。

例子 12-4 创建测试表 emp_test。

```
SQL> create table scott.emp_test
  2  as
  3  select *
  4  from scott.emp;

表已创建。
```




说明 表 emp_test 属于 SCOTT 用户，该信息也会记录在回收站中。

例子 12-5 删除表 emp_test。

```
SQL> drop table scott.emp_test;

表已删除。
```

我们模拟了一个用户错误地删除了表 EMP_TEST，如果是传统的恢复方式，则需要使用包含该表的备份文件，并且恢复过程时间相对较长，也涉及一些不必要的数据的恢复。使用了闪回技术，此时虽然删除了该表，但是该表的数据还是保留在原处，恢复时仅仅将该表从数据字典中删除记录即可。

下面查看回收站中该表的记录。注意，在回收站中记录了该表的原始名称和回收站中的名称，即对象名，并且此时需要使用具有 DBA 权限的用户登录数据库，否则会提示错误，如下例所示。

```
SQL> select owner,original_name,object_name,ts_name,droptime
  2  from dba_recyclebin;
from dba_recyclebin
     *
第 2 行出现错误:
ORA-00942: 表或视图不存在
```

下面使用 SYS 用户登录数据库，查询回收站中记录的删除表的信息，如下例所示。

例子 12-6 查看当前回收站中的记录。

```
SQL> col ts name for a10
SQL> col owner for a10
SQL> col original name for a10
SQL>select owner,original name,object name,ts name,droptime
  2* from dba recyclebin

OWNER  ORIGINAL N OBJECT NAME                    TS NAME  DROPTIME
---------- ---------- ----------------- ---------- -----------------------
SCOTT   TEST      BIN$y18lIm8wRXS9kBVBJ60uTA==$0 USERS   2010-05-22:22:49:45
SCOTT   EMP_TEST  BIN$2bqYJuA2QBG3hIePCE9Dcg==$0 USERS   2010-05-25:11:57:18
```

从输出可以看出，表 EMP_TEST 已经记录在回收站中，记录该表数据 SCOTT 用户，所属表空间为 USERS，并记录删除时间为 2010-05-25:11:57:18，注意此时自动生成了一个被删除表的系统名称为 BIN$2bqYJuA2QBG3hIePCE9Dcg==$0，该名称由 30 个字符组成。



在重命名被删除的表时，也会重命名该表的依赖对象如涉及的索引、触发器等，在恢复表时，该表涉及的依赖对象会自动恢复，但是会保留系统名称。

我们也可以使用数据字典 USER_RECYCLEBIN 以及 RECYCLEBIN 查询用户的回收站中删除的数据库对象信息。此时的 RECYCLEBIN 是 USER_RECYCLEBIN 的同义词。但是，此时必须在对象所属的用户模式下，如我们删除的表 EMP_TEST 是 SCOTT 用户拥有的表，所以必须使用 SCOTT 用户模式查询 RECYCLEBIN，如下例所示。

例子 12-7　通过 RECYCLEBIN 查询回收站中的对象记录。

```
SQL> conn scott/oracle
已连接。
SQL> show recyclebin;
ORIGINAL NAME  RECYCLEBIN NAME              OBJECT TYPE  DROP TIME
---------------- ---------------------------- ------------ -------------------
EMP_TEST       BIN$2bqYJuA2QBG3hIePCE9Dcg==$0  TABLE      2010-05-25:11:57:18
TEST           BIN$y18lIm8wRXS9kBVBJ60uTA==$0  TABLE      2010-05-22:22:49:45
```



如果使用 USER_RECYCLEBIN 查询以上输出的数据，结果相同。其实 USER_RECYCLEBIN 比 DBA_RECYCLEBIN 缺少一个 OWNER 属性，其他二者相同。

12.2.3　恢复删除的表

要删除回收站中的表，必须使用在该表所属的用户模式下执行闪回操作，如果我们删除了用户 SCOTT 的表 EMP_TEST，我们需要在 SCOTTT 用户模式下实现闪回操作。使用 FLASHBACK TABLE…… TO BEFORE DROP，其实就是要求用户具有闪回操作的权限，如具有 DROP TABLE 等权限。读者需要注意，如果操作中出现权限限制，首先需要赋予该用户相对应的操作权限，如下例所示，我们恢复删除的表 EMP_TEST。

例子 12-8　恢复删除的表 EMP_TEST。

```
SQL> flashback table emp test to before drop;

闪回完成。
```

此时，我们继续看回收站中是否还有该表记录，如下例所示。

例子 12-9　验证回收站中是否还有表 EMP_TEST 的记录。

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME          OBJECT TYPE  DROP TIME
------------- ------------------- ------------ -----------------------------
TEST            BIN$y18lIm8wRXS9kBVBJ60uTA==$0 TABLE     2010-05-22:22:49:45
```

显然，回收站中已经没有表 EMP_TEST 的记录了，我们继续查询表 EMP_TEST 的信息，如下例所示。

例子 12-10　查询用户 SCOTT 模式下的表。

```
SQL> select table name
  2  from user tables;

TABLE NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
EMP_TEST

已选择 6 行。
```

显然，表 EMP_TEST 已经重新记录在数据字典 DBA_TABLES 中了，下面继续查看表 EMP_TEST 中的数据，如下例所示。

例子 12-11　查询表 EMP_TEST 中的数据。

```
SQL> select *
  2  from emp_test
  3  where rownum<2;

     EMPNO ENAME      JOB          MGR  HIREDATE       SAL  COMM  DEPTNO
---------- -------- ----------- --------- ------------- ------ ---- -------
      7369   SMITH      CLERK       7902  17-13 月-80      800            20
```

从输出显而易见，表 EMP_TEST 的数据也已经恢复，这样就通过闪回删除技术成功恢复了用户“误删除”的表 EMP_TEST。

在闪回表时，也可以使用系统为被删除表的重命名后的名称，并且可以为恢复后的表重新命名。先查询当前用户 SCOTT 回收站中的记录，如下例所示。

例子 12-12　查询用户 SCOTT 回收站中的记录。

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                 OBJECT TYPE  DROP TIME
------------------- ------------------------- ---------- --------------------
TEST          BIN$y18lIm8wRXS9kBVBJ60uTA==$0 TABLE       2010-05-22:22:49:45
```

从输出发现表 TEST 被删除，并且被重命名为 BIN$y18lIm8wRXS9kBVBJ60uTA==$0，我们通过系统名称 BIN$y18lIm8wRXS9kBVBJ60uTA==$0 恢复表 TEST，并且重命名为 NEW_TEST，如下例所示。

例子 12-13　使用系统名称闪回表并重命名。

```
SQL> flashback table "BIN$y18lIm8wRXS9kBVBJ60uTA==$0"
  2  to before drop
  3  rename to new_test;
```

```
闪回完成。
```

提示闪回完成，接着需要验证闪回结果，注意此时已经将表 TEST 重命名为 NEW_TEST，如下例所示。

例子 12-14 验证表 NEW_TEST 是否存在。

```
SQL> select table_name
  2  from user_tables;

TABLE_NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
EMP_TEST
NEW_TEST

已选择 6 行。
```

此时，从上例的输出看出表 NET_TEST 已经重新注册到数据字典中，读者可以查询表 NEW_TEST 中的数据以验证数据是否恢复，这里不再演示。

前文讲过一旦 DROP 一个表，和表相关联的其他数据库对象如触发器、索引都将被删除，一旦闪回该表时相关的数据库对象将自动恢复，但是名称不是原先的名称，而是删除后系统自动重起的名称，该名称不易阅读，需要手工修改。下面我们演示这个过程，首先创建一个表 TEST1，并创建一个索引，然后删除表再闪回该表看看索引变化，如下例所示。

例子 12-15 创建测试表 TEST1。

```
SQL> create table test1
  2  as
  3  select *
  4  from emp;

表已创建。
```

然后，创建基于表 TEST1 的列 ENAME 的索引 SCOTT_TEST1_ENAME，如下例所示。

例子 12-16 创建索引 SCOTT_TEST1.ENAME。

```
SQL> create index scott_test1_ename on test1(ename);

索引已创建。
```

接着，验证该索引的创建结果，如下例所示。

例子 12-17 查看索引 SCOTT_TEST1.ENAME 是否创建成功。

```
SQL> select index_name,table_name
  2  from user_indexes
  3  where table_name='TEST1';
```

```
INDEX_NAME                        TABLE_NAME
------------------------------   ---------------------------
SCOTT_TEST1_ENAME                 TEST1
```

现在我们成功创建了一个表 TEST1，并基于该表的 ENAME 列创建了索引 SCOTT_TEST1_ENAME，现在先删除该表，再闪回该表，查看基于该表的索引名称的变化，如下例所示。

例子 12-18 删除表 TEST1。

```
SQL> drop table test1;

表已删除。
```

我们通过同义词 RECYCLEBIN 查看回收站中是否记录了表 TEST1 的删除记录，如下例所示。

例子 12-19 查看回收站记录。

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                  OBJECT TYPE  DROP TIME
------------- -------------------------- ------------- -------------------
TEST1          BIN$1z+DpGAGQquDxt0ygK4B3g==$0 TABLE       2010-05-25:15:34:53
```

显然，表 TEST1 的删除结果记录在回收站中。下面我们查询该表的索引 SCOTT_TEST1_ENAME 是否存在，如下例所示。

例子 12-20 查询表 TEST1 的索引。

```
SQL>  select index_name,table_name
  2   from user_indexes
  3   where table_name='TEST1';

未选定行
```

显然，此时数据字典 USER_INDEXES 已经删除了索引记录，我们继续闪回表 TEST1，如下例所示。

例子 12-21 闪回表 TEST1。

```
SQL> flashback table test1 to before drop;

闪回完成。
```

按照闪回原理，与表 TEST1 相关联的数据库对象也会自动闪回，可以通过数据字典 USER_INDEXES 查询表 TEST1 的索引是否闪回成功，如下例所示。

例子 12-22 闪回表 TEST1 后查询该表的索引闪回结果。

```
SQL> select index_name,table_name
  2  from user_indexes
  3  where table_name='TEST1';

INDEX_NAME                          TABLE_NAME
```

```
------------------------------ ------------------------------
BIN$1PD6fSonT26sNyykRvaf3g==$0    TEST1
```

我们看到表 TEST1 的索引闪回成功，但是名称依然是在闪回表时的为该索引起的名字，所以需要用户自己修改该索引的名字。

例子 12-23　重建索引 SCOTT_TEST1_ENAME。

```
SQL> drop index "BIN$1PD6fSonT26sNyykRvaf3g==$0" ;

索引已删除。
SQL> create index scott test1 ename on test1(ename);

索引已创建。
```

显然，我们是通过先删除索引 BIN$lPD6fSonT26sNyykRvaf3g==$0，然后再以例子 12-16 的方式重建索引。这样在闪回表后就重命名了闪回表的相关数据库对象。

12.2.4　应用 Purge 永久删除表

在实际的数据库维护中，用户确认不需要某个表，又出于该表的安全性考虑，不希望把它放入回收站，此时我们使用 PURGE 指令来删除该表即可，如下例所示。

例子 12-24　永久删除表。

```
SQL> drop table test1 purge;

表已删除。
```

这样删除的表不会在回收站中记录任何信息，如果此时查看回收站不会发现表 TEST1 的删除记录。

有时需要永久删除一个表空间，但是由于该表空间数据量太大，所以不希望该表空间继续占用潜在的磁盘空间，不想放入回收站，此时可以使用 DROP TABLESPACE…INCLUDING CONTENTS 指令实现，如下例所示。

例子 12-25　永久删除表空间。

```
SQL> drop tablespace test
  2  including contents;

表空间已删除。
```



上面删除了表空间 test，同时会触发一个操作，即 Oracle 会将回收站中与表空间 TEST 相关的所有表同时删除。

如果已经删除一个表，并且该表记录在回收站中，此时又希望永久删除该表，也需要使用 PURGE 指令。我们先查看当前回收站中的信息，以确认要永久删除的表，如下例所示。

例子 12-26　查看回收站中信息。

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
TEST2            BIN$NizZ+ImCQOOd7bTix4x1pA==$0 TABLE        2010-05-25:16:05:58
TEST3            BIN$8by8z0a0RHSOp6YxznMQ6Q==$0 TABLE        2010-05-25:16:06:09
```

此时，如果要永久删除表 TEST2，则使用 PURGE 指令，如下例所示。

例子 12-27　从回收站中永久删除表 TEST2。

```
SQL> purge table test2;

表已清除。
```

此时表 TEST2 将从数据库中永久删除，并且和该表对应的其他数据库对象也将永久删除。我们继续查找回收站，则不会再记录表 TEST2 的任何信息，如下例所示。

例子 12-28　查询回收站以验证表 TEST2 是否删除。

```
SQL> show recyclebin;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
TEST3            BIN$8by8z0a0RHSOp6YxznMQ6Q==$0 TABLE        2010-05-25:16:06:09
```

此时，回收站中确实已经删除了表 TEST2。

在永久删除回收站中的表时，也可以使用系统生成的名字，如下例所示。

例子 12-29　永久删除回收站中的表。

```
SQL> purge table "BIN$NizZ+ImCQOOd7bTix4x1pA==$0";

表已清除。
```

在生产数据库的维护中，我们会遇到这种情况，就是已经知道删除了某个表空间中的多个表，又打算永久删除这些表，此时如果一一删除显然会耗费时间，我们可以使用如下指令直接删除和某个表空间相关的回收站中的表。我们先查看当前回收站中的表。

例子 12-30　查看当前回收站中的表。

```
SQL> select original name,object name,ts name
  2  from user recyclebin;

ORIGINAL NAME    RECYCLEBIN NAME               TS NAME
------------------------------------------ ------------ -------------------
TEST3             BIN$8by8z0a0RHSOp6YxznMQ6Q==$0   USERS
```

当前只有表 TEST3 在表空间中，并且表 TEST3 属于 USERS 表空间，我们可以通过 PURGE 表空间的方式永久删除回收站中和表空间 USERS 相关的表，如下例所示。

例子 12-31　永久删除同一表空间中的回收站中的表。

```
SQL> purge tablespace users;
```

```
表空间已清除。
```

此时，继续查看回收站中的表记录，不会再出现与表空间 USERS 相关的表记录了，如下例所示。

例子 12-32　查看回收站验证删除表空间相关表记录结果。

```
SQL> select original name,object name,ts name
  2  from user recyclebin;
  3  where ts name='USERS';

未选定行
```

我们也可以永久删除回收站中与某个表空间相关的某个用户的表，此时除了 TABLESPACE 关键字还需要 USER 关键字，如下例所示。

例子 12-33　久删除回收站中与某个表空间相关的特定用户的表。

```
SQL> purge tablespace test user scott;

表空间已清除。
```



使用 PURGE 指令对回收站的操作，使用 DROP 指令对数据库中的表的操作，这两个操作有一点关联，但是操作的对象不同。在使用时要注意二者的操作对象。

12.3　闪回数据库

12.3.1　闪回数据库概述

闪回数据库技术是一种快速的数据库恢复方案，这种恢复是基于用户的逻辑错误，比如对表中的数据做了错误的修改、插入了大量错误数据等，此时往往需要将数据库恢复到修改之前的某个时间点。如果使用传统的恢复技术，则首先需要复原备份的数据文件，再使用归档日志恢复到以前的某个时间点，显然这样的恢复涉及很多不必要的数据恢复，并且恢复的时间主要取决于数据库的大小。这样为了“小错误”而动用全库的恢复是很耗时的行为。

Oracle 的闪回数据库技术就解决了这个问题，它使用闪回日志以及部分的归档日志来恢复用户的逻辑错误，这种恢复只针对用户错误的恢复，而不涉及整个数据库的恢复，恢复更具有针对性而且恢复时间大大减少。

闪回日志由 Oracle 自动创建，并存储在闪回恢复区中，由闪回恢复区管理。既然闪回日志由闪回恢复区管理，那么就不能保证闪回日志被保存的数据的可靠性，因为一旦快闪恢复区空间不足，会自动删除旧的闪回日志文件以腾出空间。在 Oracle 中快闪恢复区中备份文件的存储优先，所以为了保存尽可能多的闪回日志数据，最好将闪回恢复区设置得大些。

下面我们给出闪回数据库的架构图，如图 12-2 所示。

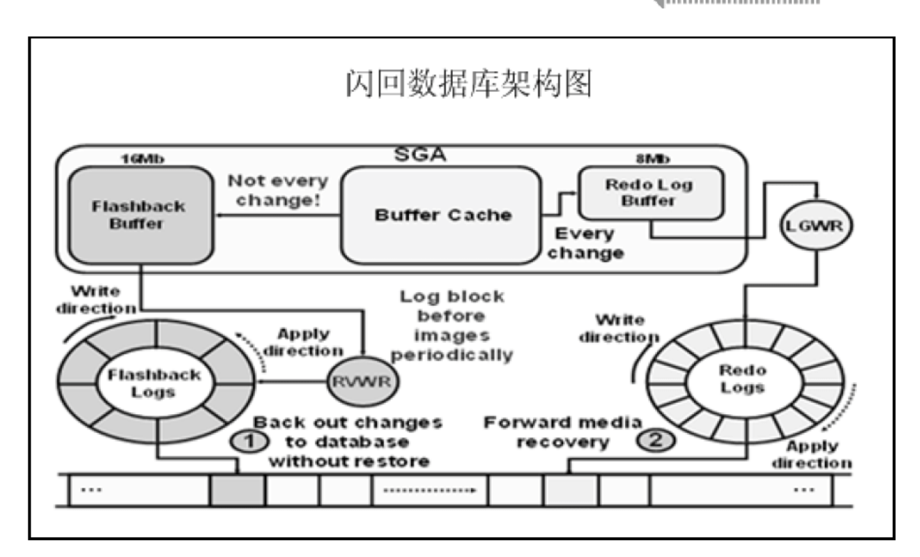


图 12-2　闪回数据库架构图

在上图闪回缓冲区中（Flashback Buffer）的变化数据将按照一定的时间间隔顺序被写入闪回日志（Flashback Logs），比如用户对表删除或者增加了数据，此时变化的前像操作就记录在闪回日志中，一旦闪回将利用相反的操作恢复修改。注意将闪回缓冲区中的数据写入闪回日志的操作由 RVWR 进程负责，一旦启动了闪回数据库，该进程会自动启动。

显然，如果数据库中绝大多数是查询操作，此时的闪回数据库开销很小，因为它不需要做什么，所以闪回数据库的系统开销取决于是否有密集的写操作。



对于闪回数据库而言，闪回日志不会被归档。

12.3.2　启用闪回数据库

Oracle 默认不启动闪回数据库，如果需要启动闪回数据库特性，必须将数据库设置为归档模式，并启用闪回恢复区，因为闪回日志文件存放在闪回恢复区中。如果在 RAC 环境下，必须将闪回恢复区存储在集群文件或 ASM 文件中。启用闪回数据库特性的操作步骤如下。

01 闪回数据库特性需要数据库处于归档模式，首先检查当前数据库的归档状态，如下例所示。

例子 12-34 查看当前数据库的归档状态。

```
SQL> archive log list;
数据库日志模式          存档模式
自动存档                启用
存档终点                USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列      3
下一个存档日志序列      5
当前日志序列            5
```

显然，当前的数据库处于归档模式，使用 DB_RECOVERY_FILE_DEST 参数指定的目录作为储存目录，该参数的值即为快速恢复区。接着可以继续查询该目录，以确定归档的操作系统存储位置。

例子 12-35 查询归档目录（快闪恢复区）。

```
SQL> show parameter db recovery file dest;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db recovery file dest                string      E:\oracle\product\10.2.0/flash
                                                  recovery area
db_recovery_file_dest_size           big integer 2G
```

快闪恢复区为 E:\oracle\product\10.2.0/flash_recovery_area，而系统为该空间分配的大小为 2G，这里我们就使用该空间作为系统的归档目录，并且闪回恢复区也是闪回数据库日志文件的存储空间。

如果数据库没有启动归档模式，则需要手工启动到归档模式。具体方法为启动数据库到 MOUNT 状态，然后使用 ALTER DATABASE ARCHIVELOG 指令启动归档模式，如下例所示。

例子 12-36 设置数据库到归档模式。

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1350380 bytes
Variable Size             136803636 bytes
Database Buffers          448790528 bytes
Redo Buffers                7135232 bytes
数据库装载完毕。
SQL> alter database archivelog;

数据库已更改。
```

02 设置参数 DB_FLASHBACK_RETENTION_TARGET，该参数的值是一个以分钟为单位的数字，默认为 1340 分钟，含义上将数据库闪回到过去的时间，即从当前开始计算最大可以把数据库闪回到过去的时间。我们可以查询该参数的设置，如下例所示。

例子 12-37 查询参数 DB_FLASHBACK_TETENTION_TARGET 的值。

```
SQL> show parameter db_flashback_retention;

NAME                                 TYPE                VALUE
------------------------------------ ------------------- ------------------------
db_flashback_retention_target        integer             1340
```

上述查询的值是系统默认的参数值，可以动态修改这个参数，使得闪回数据库可以闪回更长时间的数据，比如最长可以闪回到过去两天内的数据，即 24*60*2=2880，如下例所示。

例子 12-38 修改参数 DB_FLASHBACK_TETENTION_TARGET 的值。

```
SQL> alter system set db flashback retention target=2880 scope =both;

系统已更改。
```

参数 DB_FLASHBACK_TETENTION_TARGET 说明可以闪回数据库到过去的时间段，但是必须明确 Oracle 不保证一定可以闪回到时间段内的某个时间点，因为闪回日志是由快闪恢复区自动维护的，如果由于备份数据库而造成空间不足，此时较早的闪回日志会被删除。为了尽量避免这种情况，在系统运行一段时间，在这段时间内覆盖了数据库系统的主要工作负荷，这样就需要通过数据字典 v$flashback_database_log 来评估需要的快闪恢复区空间，如下例所示。

例子 12-39 估计当前数据库所需的快闪恢复区空间。

```
SQL> select estimated_flashback_size,retention_target,flashback_size
  2  from v$flashback_database_log;

ESTIMATED_FLASHBACK_SIZE RETENTION_TARGET FLASHBACK_SIZE
------------------------ --------- ----------------------
               107249664              1340       40386560
```

此时，参数 ESTIMATED_FLASHBACK_SIZE 说明系统估计的快闪恢复区大小为 107249664 字节。FLASHBACK_SIZE 说明当前的闪回数据的大小为 40386560 字节。

03 启动闪回数据库。要启用闪回数据库可以使用 ALTER DATABASE FLASBHBACK ON 指令，但是该指令必须在数据库处于 MOUNT 状态时运行。我们先关闭数据库并启动到 MOUNT 状态，如下例所示。

例子 12-40 关闭数据库并启动到 MOUNT 状态。

```
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。
......
数据库装载完毕。
```

将数据库启动到 MOUNT 状态后，如果此时数据库处于打开状态，而用户又试图启动闪回数据库，则会报如下错误。

```
SQL> alter database flashback on;
alter database flashback on
*
第 1 行出现错误:
ORA-38759: 数据库必须仅由一个实例装载并且不能打开。
```

接着，在数据库处于 MOUNT 状态时，启动闪回数据库，如下例所示。

例子 12-41 启动闪回数据库。

```
SQL> alter database flashback on;
```

```
数据库已更改。
```

在数据库启动了闪回特性后，可以通过数据字典视图 v$database 查看启用状态，如下例所示。

例子 12-42 查看闪回数据库启动状态。

```
SQL> select dbid,name,flashback on
  2  from v$database;

      DBID  NAME      FLASHBACK ON
----------------- ---------------- ----------------------------
2403674905  LSZPRI     YES
```

显然，在 LSZPRI 数据库上启动了闪回数据库特性，因为 FLASHBACK_ON 列的值为 YES。

12.3.3 关闭闪回数据库

默认情况下，只要启动了闪回数据库特性，数据库的永久表空间都会受闪回数据库保护，如果不希望某个表空间受闪回数据保护，可以禁用对某个表空间的闪回特性，如下例所示。

例子 12-43 禁用表空间的闪回数据库特性。

```
SQL> alter tablespace users flashback off;

表空间已更改。
```

可以通过数据字典 v$tablespace 来查询该表空间是否已经不被闪回保护，如下例所示。

例子 12-44 查询当前数据库被闪回保护状态。

```
SQL> col flashback on for a15
SQL> run
  1  select name,flashback on
  2* from v$tablespace

NAME                                     FLASHBACK ON
---------------------------------------- ------------------------------
SYSTEM                                    YES
UNDOTBS1                                 YES
SYSAUX                                    YES
USERS                                      NO
TEMP                                     YES
TEST                                       YES

已选择 6 行。
```

从输出可以看出表空间 USERS 已经不被闪回数据库保护了。如果想重新启用该表空间，使其继续启用闪回数据库特性，可以通过指令 ALTER TABLESPACE USERS FLASHBACK ON 来实现，但是必须将数据库启动到 MOUNT 状态，如下例所示。

例子 12-45　启用表空间 USERS 的闪回数据库特性。

```
SQL> startup mount;
ORACLE 例程已经启动。
..........
数据库装载完毕。
SQL> alter tablespace users flashback on;

表空间已更改。
```

此时，我们继续使用数据字典 v$tablespace 来查询当前数据库表空间被闪回数据库保护的状态，如下例所示。

例子 12-46　查看表空间被闪回数据库保护状态。

```
SQL> run
  1  select name,flashback_on
  2* from v$tablespace

NAME                                          FLASHBACK_ON
--------------------------------------------- ---------------
SYSTEM                                         YES
UNDOTBS1                                       YES
SYSAUX                                         YES
USERS                                          YES
TEMP                                           YES
TEST                                           YES

已选择 6 行。
```

注意，此时的表空间 USERS 的 FLASHBACK_ON 为 YES，说明它又重新被闪回数据库保护了。

如果需要关闭闪回数据库特性，可以关闭数据库级别的闪回数据库特性，如下例所示。

例子 12-47　关闭闪回数据库。

```
SQL> startup mount;
ORACLE 例程已经启动。
..........
数据库装载完毕。
SQL> alter database flashback off;

数据库已更改。
```

注意，上述操作必须将数据库实例启动到 MOUNT 状态，否则无法关闭或启动闪回数据库特性，并且一旦关闭闪回数据库特性，闪回日志将自动删除。

12.3.4　闪回数据库方法

闪回数据库可以使用 RMAN 方法，也可以使用 SQL 指令的方法实现，使用 RMAN 闪回数据库有如下三种方法：

- RMAN>FLASHBACK DATABASE TO TIME=TO_DATE('2010-5-25 12:43:00', 'YYYY-

MM-DD HH23:MI:SS');
该方法将数据库闪回到过去的某个时间点，通过 TO_DATE 函数指定具体的时间；

- RMAN>FLASHBACK DATABASE TO SCN=638832;
  该方法将数据库闪回到过去的某个系统 SCN，显然在实践中这个方法不容易实现，因为在数据库发生逻辑错误之前一般不会去查询数据库当前的 SCN;
- RMAN>FLASHBACK DATABASE TO SEQUENCE=345 THREAD=1;
  该方法闪回到特定日志序列号之前的状态，不包括序列号 345。

使用 SQL 指令闪回数据库有如下两种方式：

- SQL>FLASHBACK DATABASE TO TIMESTAMP(SYSDATE-1/24)
  该法方法将数据库闪回到时间戳指定的状态；
- SQL>FLASHBACK DATABASE TO SCN 678854
  该方法闪回数据库到过去的某个 SCN。

在执行闪回数据库时，需要将数据库切换到 MOUNT 状态，在闪回数据库结束后必须使用 START DATABASE OPEN RESETLOGS 打开数据库，即需要重新设置重做日志，使得重做日志序列号重新计数。

12.3.5　使用闪回数据库

本节我们使用闪回数据库到过去的某个 SCN 的方法闪回数据库。首先创建一个测试表 TEST，如下例所示。

例子 12-48　创建测试表 TEST。

```
SQL> create table test
  2  as
  3  select *
  4  from scott.emp;

表已创建。
```

此时，查看一下该表的行数，目的是以表的行数为基准记录当前的表状态，如下例所示。

例子 12-49　查询表 TEST 的行数。

```
SQL> select count(*)
  2  from test;

  COUNT(*)
----------
        13
```

当前的表有 13 行，接下来向表中插入数据，以模仿用户的逻辑错误。

在表 TEST 创建完毕后再查询当前的 SCN，记录此时的 SCN 值为闪回恢复做准备，如下例所示。

例子 12-50 查询当前的 SCN 值。

```
SQL> select current scn from v$database;

CURRENT SCN
-----------
    609842
```

当前的 SCN 为 609842，下面我们在表 TEST 中插入一些数据，模仿对表 TEST 的破坏行为，即插入了错误的数据，如下例所示。

例子 12-51 向表 TEST 中插入数据。

```
SQL> insert into test
  2  select *
  3  from scott.emp;

已创建 13 行。
```

此时，已经成功向表 TEST 插入了一些数据，但是这些数据是不希望插入的，这时需要使用闪回数据库技术闪回到插入之前的状态。再次查看当前的 SCN，如下例所示。

例子 12-52 查询当前的 SCN。

```
SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    609859
```

从输出可见当前的 SCN 为 609859，我们要恢复到之前的 SCN 为 609842 的状态。下面直接提交修改，从而实现破坏表 TEST 的目的。

```
SQL> commit;

提交完成。
```

下面的过程尝试恢复数据库到 SCN 位 609842 的状态，我们已经知道要使用闪回数据库闪回到过去某个状态，必须启动数据库到 MOUNT 状态，所以此时先关闭数据库再启动到 MOUNT 状态，如下例所示。

例子 12-53 启动数据库到 MOUNT 状态。

```
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area  603979776 bytes
Fixed Size                  1350380 bytes
Variable Size             184552372 bytes
Database Buffers          411041792 bytes
```

```
Redo Buffers              7135232 bytes
数据库装载完毕。
```

现在开始正式应用闪回数据库特性闪回到过去的 SCN 为 609842 的状态，如下例所示。

例子 12-54 闪回数据库到 SCN 为 609842 的状态。

```
SQL> flashback database to scn 609842;

闪回完成。
```

接着使用 RESETLOGS 打开数据库，如下例所示。

例子 12-55 闪回数据库后打开数据库。

```
SQL> alter database open resetlogs;

数据库已更改。
```

最后验证是否将表 TEST 恢复到以前的状态。我们查询表 TEST 的行数，如下例所示。

例子 12-56 查询表 TEST 的行数。

```
SQL> select count(*)
  2  from test;

  COUNT(*)
----------------------
        13
```

此时，表 TEST 又恢复到过去的状态，说明我们已经将数据库闪回到 SCN 为 609842 的状态了。



如果使用闪回操作造成一定的混乱，比如闪回没有达到要求，可以使用 RECOVER DATABASE 撤销闪回操作。如果闪回太多，可以使用 RECOVER DATABASE UNTIL 将数据库恢复到以前的某个时间点。

12.3.6　监控闪回数据库

本节我们介绍几个监控闪回数据库的数据字典视图，通过他们可以知道将数据库闪回到过去的时间、SCN、当前闪回日志的各种状态（如闪回起止时间、闪回记录的数据量）等信息。下面我们分别介绍这些视图。

因为无法保证一定可以闪回到从现在开始到参数 db_flashback_retention_target 指定时间段内的任意时间点，所以在闪回前最好使用数据字典 v$flashback_database_log 查询可以闪回到的最小 SCN 号以及可以闪回到的时间点，如下例所示。

例子 12-57 查询可以闪回的最小 SCN 及闪回到的时间。

```
SQL> select oldest_flashback_scn,oldest_flashback_time
  2  from v$flashback database log;

OLDEST_FLASHBACK_SCN OLDEST_FLASHBA
```

```
-------------------------------------- -----------------------------
             613813      25-5 月 -10
```

输出说明，可以将当前数据库闪回到最小的 SCN 为 613813。此时，可以继续查看系统当前的 SCN，可以预知当前 SCN 一定会大于 OLDEST_FLASHBACK_SCN 参数的值，如下例所示。

例子 12-58 查询当前数据库日志的 SCN。

```
SQL> select current scn
  2  from v$database;

CURRENT SCN
----------------------
    613155
```

显然，当前的 SCN 为 613155 比 OLDEST_FLASHBACK_SCN 参数的值要大。下面我们介绍数据字典视图 v$flashback_database_stat，该字典视图的作用是监视闪回日志写入闪回数据的各种开销，如记录了当前闪回记录起止时间、闪回记录的数据量以及重做日志记录的数据量等信息。该视图记录 24 小时内的闪回日志开销记录，每一行记录了一小时间隔的状态，如下例所示。

例子 12-59 查看当前闪回日志记录的各种开销。

```
SQL>SELECT *
  2   FROM V$FLASHBACK_DATABASE_STAT;

BEGIN_TIME END_TIME FLASHBACK_DATA   DB_DATA  REDO_DATA ESTIMATED_FLASHBACK_SIZE
------------------- ---------------- ----------- --------- -----------------
25-5 月 -10      25-5 月 -10            835584     598016     151040        0
```

其中 FLASHBACK_DATA 表示在时间间隔内记录的多少闪回数据，单位是字节参数 DB_DATA 记录了时间间隔内有多少数据块的读写，单位是数据块。参数 Redo_DATA 说明时间间隔内记录了多少重做数据，单位是字节。

我们已经反复强调过，闪回数据库不保证闪回数据到过去的某个时间点，因为闪回日志是由快闪恢复区维护的，而快闪恢复区是备份文件优先存储，即如果由于备份需要的空间不足，就会删除部分闪回日志文件。所以虽然定义了参数 db_flashback_retention_target，但是 Oracle 只是尽力保证恢复到该参数指定的、从现在开始某个时间段内的数据。因此，需要监控快闪恢复区的空间变化，在必要时候增加快闪恢复区的空间，如下例所示。

例子 12-60 查询快闪恢复区的空间使用情况。

```
SQL> select name,space limit,space used,space reclaimable,number of files
  2* from v$recovery file dest

NAME         SPACE LIMIT SPACE USED SPACE RECLAIMABLE NUMBER OF FILES
---------------- --------- --------------- -------------- ---------------
E:\oracle\product\10.2.0/flash_recovery_area 2137483648 669641728 55032832 8
```

参数 NAME 说明了快闪恢复区的目录。参数 SPACE_LIMIT 说明空间最大使用上限。参数 SPACE_USED 说明已经使用了的空间。参数 SPACE_RECLAIMABLE 说明可以回收的空间。此时需要注意已经使用的空间和总空间的值，如果二者接近就增加闪回恢复区的尺寸，或者使用其他方

式比如闪回恢复区管理等方法，防止快闪恢复区空间不足。

12.3.7　使用闪回数据库的限制

在以下几种环境下不能使用闪回数据库特性：

- 如果是数据文件被删除或缩短。
- 如果在闪回时间范围内复原或重建了一个控制文件。
- 在 RESETLOGS 操作之前。
- 表空间被删除。

在以上几种情况无法使用闪回数据库时，只能使用不完全恢复将数据库恢复到过去的某个时间点。

12.4　复原点技术

复原点顾名思义就是将数据恢复到该点时的状态，在闪回数据库或者闪回表操作时，往往需要一个具体的时间点或者 SCN 等信息 ，说明将闪回到哪里结束，而复原点就是 SCN 的别名，显然复原点更容易记忆。如下例所示，我们创建一个复原点。

例子 12-61　创建一个复原点

```
SQL> create restore point rp1;

还原点已创建。
```

通过数据字典 v$restore_point 查询刚刚创建的复原点 RP1，如下例所示。

例子 12-62　查询复原点信息。

```
SQL> select name,scn,storage_size,guarantee_flashback_database
  2  from v$restore_point;

NAME                                                   SCN  STORAGE_SIZE  GUA
--------------- ---------- ----------- ---------- ----------- --------
RP1                                                 619283             0   NO
```

可以看到此时已成功创建一个复原点，但是该复原点无非是 SCN 的别名，使用起来方便罢了，还是无法确保闪回数据一定成功，因为这依赖于需要的闪回日志数据是否存在，而闪回日志又由快闪恢复区管理。下面我们介绍一种有保证的复原点技术，这种技术可以保证在快闪恢复区空间可以保证的情况下，总可以闪回到该复原点。如果快闪恢复区空间不足，则数据库关闭。

并且使用有保证的复原点与是否使用闪回日志无关，一旦使用了有保证的复原点，Oracle 会自动保存自复原点之后的闪回日志，并不会删除这些日志。下面我们创建一个有保证的复原点，如下例所示。

例子 12-63 创建有保证的复原点。

```
SQL> create restore point grp1 guarantee flashback database;

还原点已创建。
```

在创建完有保证的复原点 grp1 后，我们再次通过数据字典视图 v$restore_point 验证是否创建成功，如下例所示。

例子 12-64 查询复原点信息。

```
SQL> select name,scn,storage_size,guarantee_flashback_database
  2  from v$restore_point;

NAME                                        SCN  STORAGE_SIZE  GUA
----------------------------------- --------- --------------------- ---
GRP1                                     619784      4096000  YES
RP1                                       619283            0  NO
```

从输出可以看出复原点 GRP1 是有保证的复原点，因为参数 guarantee_flashback_database 为 YES。并且相对于普通复原点 RP1 而言，STORAGE_SIZE 存在参数值，该参数说明为有保证的复原点指定的存储空间。

在不需要复原点时，可以通过 DROP RESTORE POINT 指令删除，删除普通复原点和有保证的复原点的操作相同，如下例所示。

例子 12-65 删除有保证的复原点。

```
SQL> drop restore point grp1;

还原点已删除。
```

此时继续查询数据字典以验证删除结果，如下例所示。

例子 12-66 查询数据字典验证删除结果。

```
SQL> select name,scn,storage size,guarantee flashback database
  2  from v$restore point;

NAME                                        SCN  STORAGE_SIZE  GUA
------------------ ---------- ------------------------------- ---
RP1                                        619283            0  NO
```

输出显示有保证的复原点 GRP1 已经被删除，当前只有一个普通复原点 RP1。

12.5　本章小结

闪回技术是 Oracle 10g 以后的版本增加的新特性。使用闪回特性可以快速的恢复用户的逻辑错误或者误删除表等操作。闪回删除主要是关注用户误删除表、索引的数据库对象，闪回删除使用回收站作为存储删除的数据库对象的逻辑存储结构，一定要理解闪回删除并不是直接将数据从数据库中删除放入回收站，而只是将数据库对象的定义从数据字典中删除，数据存储在原处。在回收站

中记录这个被删除的数据库对象，一旦需要闪回到删除前的状态，使用回收站中记录的对象信息进行闪回。在删除表时，与表相关联的其他数据库对象如索引、触发器等也都一并删除。在闪回表时，这些相关的数据库对象被自动恢复，但是名称需要进一步修改。

闪回数据库技术是处理用户逻辑错误的快速数据恢复技术，相比传统的数据库恢复会减少恢复时间，传统的数据库恢复时间取决于数据库自身的大小，而闪回数据库技术的恢复时间取决于逻辑错误的数据量大小。使用闪回数据库技术需要将数据库设置为归档模式，并启动快闪恢复区作为闪回日志的存储目录。一般条件下应该尽量设置较大的快闪恢复区，以保证闪回数据库总可以执行成功。